

## Project directory structure

---

<b>OptiWay/</b>	
<b>assets/</b>	assets (YAML, SVG, etc.) used in the program
└ (24 files and directories)	
<b>multi_agent_path_finding/</b>	C++ pathfinding algorithms
├ floyd.cpp	shortest path algorithm
├ json.cpp	(dependency) JSON parsing library
└ multi-objective-agent.cpp	path optimization algorithm
<b>optiway/</b>	program entry point; Rust project root folder
└ assets/	assets (YAML, SVG, etc.) used in the program
└ (3 files and directories)	
└ bin/	final binaries produced and final output
└ (8 files)	
└ src/	main source code
└ md_icons/	(dependency) icon packs
└ material_design_icons.rs	
└ material_symbols.rs	
└ mod.rs	
├ app_init.rs	application initialization configurations
├ app.rs	main program and UI
├ lib.rs	project structure configuration
└ main.rs	program entry point
└ target/	intermediate files during compilation
└ (100+ files and directories)	
└ bundle/	final output bundle
└ (3 files and directories)	
└ Cargo.lock	auto-generated dependency configuration
└ build.rs	Rust project build script
└ Cargo.toml	Rust project configuration
└ Makefile.toml	compilation rules and scripts
└ rustfmt.toml	Rust code formatter configuration
<b>scripts/</b>	Python scripts to aid development
└ coord-flatten.py	flattens a specific YAML file
└ coord-generator.py	generates x-coordinates of specific nodes
└ crossing-paths.py	crossing paths generation helper
└ path-converter.py	converts YAML path data to TXT
└ path-generator.py	generates paths on the same floor on one block
<b>timetable_generation</b>	Python script to generate test data
└ return-structure.txt	notes on return structure
└ timetable_generator.py	Python script to generate test data
.gitignore	Git ignored files configuration
TECHNICAL_NOTES.md	technical notes used in development

---

It should be noted that only files that are one of the following formats are included in this document, as these are the only code files:

- C++
- Rust
- TOML
- Python

Only underlined files are included as they are the only code-related files.

The source code can be found at <https://github.com/micfong-z/OptiWay/> (limited access before project due date).

The project is fully tested using the following system environment:

### Operating System

macOS Sonoma 14.0

### Rust

**Compiler:** rustc 1.75.0-nightly (2e5a9dd6c 2023-10-02)

**Toolchain:** nightly-aarch64-apple-darwin

**Cargo components:**

  cargo 1.75.0-nightly (59596f0f3 2023-09-29)

  cargo-make 0.36.12

  clippy 0.1.74 (2e5a9dd6 2023-10-02)

**Formatter:** rustfmt 1.6.0-nightly (2e5a9dd6 2023-10-02)

### C++

**Compiler:** Apple clang version 15.0.0 (clang-1500.0.40.1)

**Standard:** C++17

**Formatter:** clang-format

### Python

**Version:** Python 3.11.5

**Formatter:** autopep8 2.0.2

Note that it is not possible to compile this program only with the code included in this document due to missing assets and dependencies. Due to the extremely large file sizes, they are not shown in this document. For a complete source code repository, see <https://github.com/micfong-z/OptiWay/> (limited access before project due date).

## multi\_agent\_path\_finding/floyd.cpp

```

56  /* Initialize the dist and predecessor's matrices for floyd */
57  const int INF = numeric_limits<int>::max();
58
59  for (const auto& [node, __] : graph) {
60      for (const auto& [node2, __] : graph) {
61          if (node == node2) {
62              dist[node][node2] = 0;
63          } else {
64              dist[node][node2] = INF;
65          }
66          pred[node][node2] = "";
67      }
68  }
69
70  for (const auto& [node, edges] : graph) {
71      for (const auto& edge : edges) {
72          dist[node][edge.dest] = edge.weight;
73          pred[node][edge.dest] = node;
74      }
75  }
76 }
77
78 string concatenate(const vector<string>& vec) {
79     string result;
80     for (const auto& str : vec) {
81         if (!result.empty()) {
82             result += " "; // Add space before appending, but not for the first
83             string
84         }
85         result += str;
86     }
87     return result;
88 }
89
90 void floydWarshall(const Graph& graph, DistanceMatrix& dist, PredecessorMatrix&
91 pred) {
92     /* Main Floyd's program */
93     const int INF = numeric_limits<int>::max();
94
95     for (const auto& [k, __] : graph) {
96         for (const auto& [i, __] : graph) {
97             for (const auto& [j, ___] : graph) {
98                 if (dist[i][k] != INF && dist[k][j] != INF &&
99                     dist[i][k] + dist[k][j] < dist[i][j]) {
100                     dist[i][j] = dist[i][k] + dist[k][j];
101                     pred[i][j] = pred[k][j];
102                 }
103             }
104         }
105     }
106
107     vector<string> reconstructPath(const PredecessorMatrix& pred, const string& start,
108                                     const string& end) {
109         /* Reconstruct the path from the predecessor's list */
110         vector<string> path;
111
112         if (pred.at(start).at(end) == "") {
113             return path; // Empty path means no path exists

```

```

113     }
114
115     string at = end;
116     while (at != start) {
117         path.push_back(at);
118         at = pred.at(start).at(at);
119     }
120     path.push_back(start);
121     reverse(path.begin(), path.end());
122     return path;
123 }
124
125 json getRoutesfromTimetable(const json& timetables, const Graph& graph,
126                             const json& shortest_paths) {
127     /* Obtain the routes as a json file from the timetables's json*/
128     json routes;
129
130     for (const auto& [student, timetable] : timetables.items()) {
131         json week;
132         for (const auto& [day, classes] : timetable.items()) {
133             json today;
134
135             // Start of the day, going from G_floor to the first class
136             string path_str = shortest_paths["G" + classes["1"].get<string>()];
137             today["0"] = path_str;
138
139             // Lunch time is denoted by period "6", and periods after are delayed
140             // by one
141             // index
142             path_str =
143                 shortest_paths["G" +
144                               classes["7"].get<string>()]; // Originally, P7 is
145             the
146             // first afternoon
147             class
148             today["7"] = path_str;
149
150             int offset = 0;
151             for (int period = 1; period <= 11; period++) {
152                 // Skip P7 as it's already been handled
153                 if (period == 7) {
154                     offset = 1; // Add offset to be minused because P7 is now P6
155                     continue;
156                 }
157
158                 const string& current_period = to_string(period - offset);
159                 const string& current_room = classes[current_period];
160
161                 // cout << current_room << endl;
162
163                 if (current_room[0] != 'A' && current_room[0] != 'B' &&
164                     current_room[0] != 'G')
165                     continue;
166                 // If not the last class
167                 if (period != 11) {
168                     // If AS/AL students have P6s, they should go to G_floor in the
169                     end
170                     if (period == 6 && stoi(student) < 22000) {
171                         if (current_room == "G") {

```

```

168             today["6"] = "";
169             continue;
170         }
171         string path_str = shortest_paths[current_room + "G"];
172         today["6"] = path_str;
173         continue;
174     }
175
176     if (period == 6 && stoi(student) >= 22000) {
177         today["6"] = "";
178         continue;
179     }
180
181     const string& next_period = to_string(period - offset + 1);
182     const string& next_room = classes[next_period];
183
184     if (next_room[0] != 'A' && next_room[0] != 'B' && next_room[0]
185     != 'G')
186         continue;
187
188     if (next_room == current_room)
189         today[to_string(period)] = "";
190     else {
191         string path_str = shortest_paths[current_room + next_room]
192         ;
193         today[to_string(period)] = path_str;
194     }
195 } else {
196     string path_str = shortest_paths[current_room + "G"];
197     today[to_string(period)] = path_str;
198 }
199
200 week[day] = today;
201 }
202 routes[student] = week;
203 }
204
205 return routes;
206 }
207
208 // pre-calculated all the shortest paths in advance to avoid run-time calculations
209 void generate_all_paths(const Graph& graph) {
210     DistanceMatrix dist;
211     PredecessorMatrix pred;
212
213     initializeDistanceAndPredecessorMatrices(graph, dist, pred);
214     floydWarshall(graph, dist, pred);
215
216     json routes;
217     for (const auto [room1, _] : graph) {
218         for (const auto [room2, _] : graph) {
219             if (room1[0] != 'A' && room1[0] != 'B' && room1[0] != 'G') continue;
220             if (room2[0] != 'A' && room2[0] != 'B' && room2[0] != 'G') continue;
221             if (room1 == room2) continue;
222             vector<string> path = reconstructPath(pred, room1, room2);
223             string path_str = concatenate(path);
224             routes[room1 + room2] = path_str;
225         }
226     }

```

```

225
226     std::ofstream file("shortest_paths.json");
227     if (file.is_open()) {
228         file << routes.dump(4); // 4 spaces for indentation
229         file.close();
230         std::cout << "JSON data written to routes.json successfully." << endl;
231     } else {
232         std::cerr << "Failed to open the file for writing." << endl;
233     }
234 }
235
236 void generate_all_floyd_distances(const Graph& graph) {
237     DistanceMatrix dist;
238     PredecessorMatrix pred;
239
240     initializeDistanceAndPredecessorMatrices(graph, dist, pred);
241     floydWarshall(graph, dist, pred);
242
243     json distances;
244     for (const auto [room1, _] : graph) {
245         for (const auto [room2, __] : graph) {
246             if (room1[0] != 'A' && room1[0] != 'B' && room1[0] != 'G') continue;
247             if (room2[0] != 'A' && room2[0] != 'B' && room2[0] != 'G') continue;
248             if (room1 == room2) continue;
249             distances[room1 + room2] = dist[room1][room2];
250         }
251     }
252
253     std::ofstream file("../assets/distances.json");
254     if (file.is_open()) {
255         file << distances.dump(4); // 4 spaces for indentation
256         file.close();
257         std::cout << "JSON data written to distances.json successfully." << endl;
258     } else {
259         std::cerr << "Failed to open the file for writing." << endl;
260     }
261 }
262
263 int main() {
264     string timetable_path;
265
266     ios::sync_with_stdio(false);
267     cin.tie(0);
268
269     // general input: the path to the time table json
270     // cin >> timetable_path; // "../assets/timetable_0.json"
271     getline(cin, timetable_path);
272
273     // parse the JSON file
274     ifstream input_file(timetable_path);
275     if (!input_file.is_open()) {
276         cerr << "Failed to open the timetable file." << endl;
277         return 1;
278     }
279
280     json timetables;
281     input_file >> timetables;
282     input_file.close();

```

```
283
284 Graph graph = createSchoolGraph("../assets/paths.txt");
285 // generate_all_paths(graph);
286
287 ifstream paths_file("../assets/shortest_paths.json");
288 if (!paths_file.is_open()) {
289     cerr << "Failed to open shortest_paths.json" << endl;
290     return 1;
291 }
292
293 json shortest_paths;
294 paths_file >> shortest_paths;
295 paths_file.close();
296
297 json routes = getRoutesfromTimetable(timetables, graph, shortest_paths);
298
299 ofstream file("routes.json");
300 if (file.is_open()) {
301     file << routes.dump(); // faster i/o
302     file.close();
303     cout << "JSON data written to routes.json successfully." << endl;
304 } else {
305     cerr << "Failed to open the file for writing." << endl;
306 }
307
308 return 0;
309 }
310 }
```

## multi\_agent\_path\_finding/multi-objective-agent.cpp

```
1 #include <climits>
2 #include <cmath>
3 #include <fstream>
4 #include <iostream>
5 #include <queue>
6 #include <sstream>
7 #include <string>
8 #include <unordered_map>
9 #include <vector>
10
11 #include "json.hpp"
12 using json = nlohmann::json;
13
14 using namespace std;
15
16 // Edge for the school's graph
17 struct Edge {
18     string dest;    // destination node
19     int weight;    // edge weight (in this case, distance)
20     int type;      // edge type
21 };
22 // Struct to store the student's path at a period
23 struct StudentPath {
24     string id;
25     double rperf;
26     vector<string> path;
27 };
28 // Override the comparison for the priority queue
29 struct CompareStudentPath {
30     bool operator()(const StudentPath& s1, const StudentPath& s2) {
31         return s1.rperf < s2.rperf;
32     }
33 };
34
35 using Graph = unordered_map<string, vector<Edge>>;
36 using PathPQ = priority_queue<StudentPath, vector<StudentPath>, CompareStudentPath>;
37
38 int BATCH_SIZE = 10;
39 const double CONGESTION_PENALTY = 10000.0;
40 int ITER_NUM = INT_MAX;
41 int ITER_COUNT = 1;
42 string ROUTE_FILE_PATH;
43 int ITER_SAVE_STEPS = 500;
44
45 // Code to create the school's layout graph from the path.txt
46 Graph createSchoolGraph(const string& file_path) {
47     Graph graph;
48     ifstream file(file_path);
49
50     if (!file.is_open()) {
51         cerr << "Failed to open the file." << endl;
52         return graph;
53     }
54
55     string line;
```

```

56     while (getline(file, line)) {
57         istringstream iss(line);
58
59         string node1, node2;
60         int distance, type;
61
62         iss >> node1 >> node2 >> distance >> type;
63
64         Edge edge1 = {node2, distance, type};
65         Edge edge2 = {node1, distance, type};
66
67         graph[node1].push_back(edge1);
68         graph[node2].push_back(edge2); // Because it's an undirected graph
69     }
70
71     file.close();
72     return graph;
73 }
74
75 // Codes for getting Floyd's shortest path for all the students
76 json getRoutesfromTimetable(const json& timetables, const Graph& graph,
77                             const json& shortest_paths) {
78     /* Obtain the routes as a json file from the timetables's json*/
79     json routes;
80
81     for (const auto& [student, timetable] : timetables.items()) {
82         json week;
83         for (const auto& [day, classes] : timetable.items()) {
84             json today;
85
86             // Start of the day, going from G_floor to the first class
87             string path_str = shortest_paths["G" + classes["1"].get<string>()];
88             today["0"] = path_str;
89
90             // Lunch time is denoted by period "6", and periods after are delayed
91             // by one
92             // index
93             path_str =
94                 shortest_paths["G" +
95                               classes["7"].get<string>()]; // Originally, P7 is
96             the
97             // first afternoon
98             class
99             today["7"] = path_str;
100
101            int offset = 0;
102            for (int period = 1; period <= 11; period++) {
103                // Skip P7 as it's already been handled
104                if (period == 7) {
105                    offset = 1; // Add offset to be minused because P7 is now P6
106                    continue;
107                }
108
109                const string& current_period = to_string(period - offset);
110                const string& current_room = classes[current_period];
111
112                // cout << current_room << endl;
113
114                if (current_room[0] != 'A' && current_room[0] != 'B' &&

```

```

112         current_room[0] != 'G')
113     continue;
114 // If not the last class
115 if (period != 11) {
116     // If AS/AL students have P6s, they should go to G_floor in the
117     end
118     if (period == 6 && stoi(student) < 22000) {
119         if (current_room == "G") {
120             today["6"] = "";
121             continue;
122         }
123         string path_str = shortest_paths[current_room + "G"];
124         today["6"] = path_str;
125         continue;
126     }
127
128     if (period == 6 && stoi(student) >= 22000) {
129         today["6"] = "";
130         continue;
131     }
132
133     const string& next_period = to_string(period - offset + 1);
134     const string& next_room = classes[next_period];
135
136     if (next_room[0] != 'A' && next_room[0] != 'B' && next_room[0]
137         != 'G')
138         continue;
139
140         if (next_room == current_room)
141             today[to_string(period)] = "";
142         else {
143             string path_str = shortest_paths[current_room + next_room];
144             today[to_string(period)] = path_str;
145         }
146     } else {
147         string path_str = shortest_paths[current_room + "G"];
148         today[to_string(period)] = path_str;
149     }
150     week[day] = today;
151 }
152 routes[student] = week;
153
154 return routes;
155 }
156
157 // Code to slice a route string into a vector of strings
158 vector<string> vectorizeString(const string s) {
159     vector<string> tokens;
160     int start = 0;
161     int end = s.find(' ');
162
163     while (end != string::npos) {
164         tokens.push_back(s.substr(start, end - start));
165         start = end + 1;
166         end = s.find(' ', start);
167     }

```

```

168     tokens.push_back(s.substr(start, end));
169
170     return tokens;
171 }
172
173 // Code to concatenate a vector of strings into a single string
174 string concatenate(const vector<string>& vec) {
175     string result;
176     for (const auto& str : vec) {
177         if (!result.empty()) {
178             result += " "; // Add space before appending, but not for the first
179             string
180         }
181         result += str;
182     }
183     return result;
184 }
185
186 // Code to get the shortest path from start to end using Dijkstra's algorithm with
187 // congestion penalty
188 vector<string> getDijkstraPenaltiedPath(const string& start, const string& end,
189                                         const Graph& graph,
190                                         const unordered_map<string, int>&
191                                         congestion) {
192     unordered_map<string, double> distances;
193     unordered_map<string, bool> visited;
194     unordered_map<string, string> prev; // to store the path
195
196     vector<string> path;
197     if (start == end) return path = {"G", "G"};
198
199     for (const auto& [node, _] : graph) {
200         distances[node] = INT_MAX;
201         visited[node] = false;
202     }
203     distances[start] = 0;
204
205     priority_queue<pair<double, string>> pq;
206     pq.push({0, start});
207
208     while (!pq.empty()) {
209         string current = pq.top().second;
210         pq.pop();
211
212         if (visited[current]) continue;
213         visited[current] = true;
214
215         string destination;
216         try {
217             for (const auto& edge : graph.at(current)) {
218                 destination = edge.dest;
219                 double edgeWeight = edge.weight + (CONGESTION_PENALTY *
220                                                 congestion.at(current +
221                                                 edge.dest));
222                 if (!visited[edge.dest] &&
223                     distances[current] + edgeWeight < distances[edge.dest]) {
224                     distances[edge.dest] = distances[current] + edgeWeight;
225                     pq.push({-distances[edge.dest], edge.dest});
226                 }
227             }
228         } catch (...) {
229             cerr << "Error: Failed to access graph at index " << current << endl;
230         }
231     }
232
233     if (end == "G") {
234         path = {"G"};
235     } else {
236         string dest = end;
237         while (prev[dest] != start) {
238             dest = prev[dest];
239             path.push_back(dest);
240         }
241         reverse(path.begin(), path.end());
242     }
243
244     return path;
245 }

```

```

224         prev[edge.dest] =
225             current; // save the previous node for path
226     }
227     }
228 } catch (const out_of_range& e) {
229     cout << current << endl;
230     cerr << "Error: " << e.what() << endl;
231 }
232 }
233
234 // Reconstruct path from 'end' to 'start' using the 'prev' mapping
235 for (string at = end; at != ""; at = prev[at]) {
236     path.push_back(at);
237 }
238 reverse(path.begin(), path.end()); // since the path is constructed in
reverse
239
240     return path;
241 }
242
243 // Code to calculate r_perf using a whole congestion matrix
244 double computePerformanceIndex(const vector<string>& route,
245                                 const unordered_map<string, int>& congestion,
246                                 const Graph& graph) {
247     double r_perf = 0.0;
248     int c = 0;
249
250     if (route[0] == "G" && route[1] == "G") return 0.0; // spare class
251
252     for (int i = 0; i < route.size() - 1; i++) {
253         string start = route[i];
254         string end = route[i + 1];
255
256         if (start == "G" || end == "G") continue; // Ignore the paths to and from
G_floor
257
258         int w_i;
259         for (const auto& edge : graph.at(start)) {
260             if (edge.dest == end) {
261                 w_i = edge.weight;
262                 break;
263             }
264         }
265
266         int c_i = congestion.at(start + end);
267         r_perf +=
268             w_i * (2 + (exp((c_i - 300.0) / 200.0) - exp(-(c_i - 300.0) / 200.0))
269             / (exp((c_i - 300.0) / 200.0) + exp(-(c_i - 300.0) /
200.0)));
270         c += c_i;
271     }
272
273     return r_perf;
274 }
275
276 // Code for a single iteration of the code for a single period
277 void iter(PathPQ& paths, double& sum_rperf, unordered_map<string, int>&
congestion,

```

```

278         const Graph& graph, string& last_start, string& last_end,
279         vector<StudentPath>& temp) {
280     // A single iteration
281     StudentPath worst_path;
282     bool flag = false;
283     while (!paths.empty()) {
284         worst_path = paths.top(); // The student whose route has the greatest
285         r_perf
286         paths.pop();
287         if (worst_path.path[0] == last_start &&
288             worst_path.path[worst_path.path.size() - 1] == last_end) {
289             temp.push_back(worst_path);
290         } else {
291             flag = true;
292             break;
293         }
294     }
295     if (!flag) return;
296     sum_rperf -= worst_path.rperf;
297     vector<string> new_route = getDijkstraPenaltiedPath(
298         worst_path.path[0], worst_path.path[worst_path.path.size() - 1], graph,
299         congestion);
300     // for (const auto v: worst_path.path) cout << v << " ";
301     // cout << endl;
302     // for (const auto v: new_route) cout << v << " ";
303     double new_rperf = computePerformanceIndex(new_route, congestion, graph);
304     // cout << worst_path.rperf << " " << new_rperf << endl;
305     if (new_rperf < worst_path.rperf) {
306         StudentPath new_path = {worst_path.id, new_rperf, new_route};
307         paths.push(new_path);
308         sum_rperf += new_rperf;
309         // cout << "UPDATED PATH " << worst_path.id << endl;
310     } else {
311         // if the path doesn't change, the student's path cannot be further
312         // optimized as
313         // the congestion penalty is already too high
314         temp.push_back(worst_path);
315         sum_rperf += worst_path.rperf;
316         last_start = worst_path.path[0],
317         last_end = worst_path.path[worst_path.path.size() - 1];
318     }
319     // TODO: validate last_start and last_end
320 }
321
322 // Code for multiple iterations of the code for a single period
323 void iterMultiple(PathPQ& paths, double& sum_rperf,
324                     unordered_map<string, int>& congestion, const Graph& graph) {
325     string last_start, last_end;
326     vector<StudentPath> temp;
327     PathPQ paths_copy = paths;
328     double sum_rperf_copy = sum_rperf;
329     for (int i = ITER_COUNT; i < ITER_NUM; i++) {
330         // A single iteration
331         iter(paths, sum_rperf, congestion, graph, last_start, last_end, temp);
332
333         if (i % BATCH_SIZE | (i == 0))
334             cout << "ITER " << i << " APPR " << sum_rperf << endl;
335         else { // When the batch size is met
336             for (const auto& [node, edges] : graph) {

```

```

335         for (const auto& edge : edges) {
336             congestion[node + edge.dest] = 0;
337         }
338     }
339
340     vector<StudentPath>
341     all_paths; // contain all the paths from paths pq and temp vector
342     PathPQ new_paths;
343     // Get the congestion for the period
344     while (!paths.empty()) {
345         StudentPath path = paths.top();
346         vector<string> route = path.path;
347         paths.pop();
348         for (int i = 0; i < route.size() - 1; i++)
349             congestion[route[i] + route[i + 1]]++;
350         new_paths.push(path);
351         all_paths.push_back(path);
352     }
353
354     for (const StudentPath& path : temp) {
355         vector<string> route = path.path;
356         for (int i = 0; i < route.size() - 1; i++)
357             congestion[route[i] + route[i + 1]]++;
358         all_paths.push_back(path);
359     }
360
361     sum_rperf = 0.0;
362     for (const StudentPath& path : all_paths) {
363         double rperf = computePerformanceIndex(path.path, congestion,
364                                               graph);
365         sum_rperf += rperf;
366     }
367
368     paths = new_paths;
369
370     if (sum_rperf < sum_rperf_copy) {
371         sum_rperf_copy = sum_rperf;
372         paths_copy = paths;
373     } else {
374         temp.push_back(paths_copy.top());
375         paths_copy.pop();
376         paths = paths_copy;
377         sum_rperf = sum_rperf_copy;
378     }
379
380     cout << "ITER " << i << " ACC" << sum_rperf << endl;
381 }
382
383 return;
384
385 // Code for generating the route for a single period
386 void iterSinglePeriod(const int day, const int period, json& route_tables,
387                       const Graph& graph) {
388     // route_tables is the floyd_marshall's routes for all students at all periods
389     // generated from gerRoutesfromTimetable() Initialize the congestion matrix
390     unordered_map<string, int> congestion;
391     for (const auto& [node, edges] : graph) {
392         for (const auto& edge : edges) {

```

```

393         congestion[node + edge.dest] = 0;
394     }
395 }
396
397 // Get the congestion for the period
398 for (const auto& [student, route_table] : route_tables.items()) {
399     vector<string> route =
400         vectorizeString(route_table[to_string(day)][to_string(period)].get<
401 string>());
402     for (int i = 0; i < route.size() - 1; i++) {
403         congestion[route[i] + route[i + 1]]++;
404         congestion[route[i + 1] + route[i]]++;
405     }
406 }
407
408 // Update the pq of paths
409 PathPQ paths;
410 double sum_rperf = 0;
411 for (const auto& [student, route_table] : route_tables.items()) {
412     vector<string> route =
413         vectorizeString(route_table[to_string(day)][to_string(period)].get<
414 string>());
415     double rperf = computePerformanceIndex(route, congestion, graph);
416     StudentPath path = {student, rperf, route};
417     paths.push(path);
418     sum_rperf += rperf;
419 }
420 cout << "INITIAL PERFORMANCE: " << sum_rperf << endl;
421
422 // Run the iterations
423 iterMultiple(paths, sum_rperf, congestion, graph);
424
425 // Save the routes
426 while (!paths.empty()) {
427     StudentPath path = paths.top();
428     paths.pop();
429     route_tables[path.id][to_string(day)][to_string(period)] =
430 concatenate(path.path);
431 }
432
433 cout << "FINAL PERFORMANCE: " << sum_rperf << endl;
434
435 // Code for generating the route for a single day, with each iter covering all
436 // periods
437 void iterSingleDay(const int day, json& route_tables, const Graph& graph,
438                     json& perf_indices) {
439     PathPQ paths[15], paths_copy[15];
440     double sum_rperf[15] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
441             sum_rperf_copy[15];
442     unordered_map<string, int> congestions[15];
443     double prev_best_rperf[15];
444
445     for (int period = 0; period <= 11; period++) {
446         if (period == 1 || period == 3 || period == 8 || period == 10)
447             continue; // All the students are having classes right now
448     }

```

```

448     for (const auto& [node, edges] : graph) {
449         for (const auto& edge : edges) {
450             congestions[period][node + edge.dest] = 0;
451         }
452     }
453
454     // Get the congestion for the period
455     for (const auto& [student, route_table] : route_tables.items()) {
456         const string route_str = route_table[to_string(day)][to_string(period)]
457     ];
458         if (route_str == "") continue;
459         vector<string> route = vectorizeString(route_str);
460         for (int i = 0; i < route.size() - 1; i++) {
461             congestions[period][route[i] + route[i + 1]]++;
462             congestions[period][route[i + 1] + route[i]]++;
463         }
464     }
465
466     for (const auto& [student, route_table] : route_tables.items()) {
467         const string route_str = route_table[to_string(day)][to_string(period)]
468     ];
469         vector<string> route;
470         if (route_str == "") {
471             route = {"G", "G"}; // if this student is having a spare class
472         } else {
473             route = vectorizeString(route_str);
474         }
475         double rperf = computePerformanceIndex(route, congestions[period],
476 graph);
477         StudentPath path = {student, rperf, route};
478         paths[period].push(path);
479         sum_rperf[period] += rperf;
480     }
481
482     paths_copy[period] = paths[period];
483     sum_rperf_copy[period] = sum_rperf[period];
484     prev_best_rperf[period] = sum_rperf[period];
485
486     // cout << "PERIOD " << period << " INITIAL PERFORMANCE: " <<
487     sum_rperf[period] <<
488         // endl;
489     }
490
491     string last_start[15], last_end[15];
492     vector<StudentPath> temp[15];
493     for (int i = ITER_COUNT + 1; i <= ITER_NUM; i++) {
494         for (int period = 0; period <= 11; period++) {
495             if (period == 1 || period == 3 || period == 8 || period == 10)
496                 continue; // All the students are having classes right now
497
498             iter(paths[period], sum_rperf[period], congestions[period], graph,
499                  last_start[period], last_end[period], temp[period]);
500             prev_best_rperf[period] = min(sum_rperf[period],
501 prev_best_rperf[period]);
502
503             if (i % BATCH_SIZE | (i == 0)) {
504                 // cout << fixed << setprecision(0) << "0 " << i << ' ' << day <<
505                 ' ' <<
506                 // period << ' ' << sum_rperf[period] << ' ' <<
507                 prev_best_rperf[period] <<
508                 // endl;

```

```

501     } else {
502         for (const auto& [node, edges] : graph) {
503             for (const auto& edge : edges) {
504                 congestions[period][node + edge.dest] = 0;
505             }
506         }
507     }
508
509     vector<StudentPath>
510     all_paths; // contain all the paths from paths pq and temp
511
512     PathPQ new_paths;
513     // Get the congestion for the period
514     while (!paths[period].empty()) {
515         StudentPath path = paths[period].top();
516         vector<string> route = path.path;
517         paths[period].pop();
518         for (int i = 0; i < route.size() - 1; i++)
519             congestions[period][route[i] + route[i + 1]]++;
520         new_paths.push(path);
521         all_paths.push_back(path);
522     }
523
524     for (const StudentPath& path : temp[period]) {
525         vector<string> route = path.path;
526         for (int i = 0; i < route.size() - 1; i++)
527             congestions[period][route[i] + route[i + 1]]++;
528         all_paths.push_back(path);
529     }
530
531     sum_rperf[period] = 0.0;
532     for (const StudentPath& path : all_paths) {
533         double rperf =
534             computePerformanceIndex(path.path, congestions[period],
535                                     graph);
536         sum_rperf[period] += rperf;
537     }
538
539     paths[period] = new_paths;
540
541     if (sum_rperf[period] > sum_rperf_copy[period]) {
542         temp[period].push_back(paths_copy[period].top());
543         paths_copy[period].pop();
544         paths[period] = paths_copy[period];
545         sum_rperf[period] = sum_rperf_copy[period];
546     } else {
547         paths_copy[period] = paths[period];
548         sum_rperf_copy[period] = sum_rperf[period];
549     }
550     prev_best_rperf[period] = min(sum_rperf[period],
551                                   prev_best_rperf[period]);
552     perf_indices[to_string(day)][to_string(period)] =
553     int(sum_rperf[period]);
554     cout << fixed << setprecision(0) << "0 " << i << ' ' << day << ' '
555     << period << ' ' << sum_rperf[period] << ' '
556     << prev_best_rperf[period] << endl;
557 }
558
559 if (!(i % ITER_SAVE_STEPS) && period == 0 &&
560      i) { // for every 500 iterations dump the json file

```

```

556         vector<pair<string, string>>
557             all_paths; // contain the students and their related paths
558
559     while (!paths[period].empty()) {
560         StudentPath path = paths[period].top();
561         paths[period].pop();
562         all_paths.push_back({path.id, concatenate(path.path)} );
563     }
564
565     for (const StudentPath& path : temp[period])
566         all_paths.push_back({path.id, concatenate(path.path)} );
567
568     for (const auto& path : all_paths)
569         route_tables[path.first][to_string(day)][to_string(period)] =
570             path.second;
571
572     json iter_output;
573     iter_output["iter"] = i;
574     iter_output["indices"] = perf_indices;
575     iter_output["routes"] = route_tables;
576
577     ofstream iter_output_file(ROUTE_FILE_PATH + "_" + to_string(day) +
578                               ".json");
579     if (iter_output_file.is_open()) {
580         iter_output_file << iter_output.dump(); // minimize size
581         iter_output_file.close();
582         // cout << "JSON data written to routes.json successfully." <<
583         endl;
584     }
585     cout << fixed << setprecision(0) << "1 " << i << ' ' << day <<
586     << period << ' ' << sum_rperf[period] << ' '
587     << prev_best_rperf[period] << endl;
588 } else {
589     // cerr << "Failed to open the file for writing." << endl;
590     cout << fixed << setprecision(0) << "!" << i << ' ' << day <<
591     << period << ' ' << sum_rperf[period] << ' '
592     << prev_best_rperf[period] << endl;
593 }
594 }
595
596 int main(int argc, char** argv) {
597     // args parsing
598     int day;
599     for (int i = 0; i < argc; ++i) {
600         if (strcmp(argv[i], "-b") == 0 &&
601             i + 1 < argc) { // Batch Size: the amount of iterations before
602             updating the
603                         // congestion
604             BATCH_SIZE = stoi(argv[i + 1]);
605         }
606         if (strcmp(argv[i], "-f") == 0 &&
607             i + 1 < argc) { // Route File Path: the path to the routes file
608             ROUTE_FILE_PATH = argv[i + 1];
609         }
610         if (strcmp(argv[i], "-d") == 0 &&

```

```
611         i + 1 < argc) { // Day: the day to run the algorithm for
612             day = stoi(argv[i + 1]);
613         }
614         if (strcmp(argv[i], "-s") == 0 &&
615             i + 1 < argc) { // Save Iteration Step: the amount of iterations to
before                         // each save
616             ITER_SAVE_STEPS = stoi(argv[i + 1]);
617         }
618     }
619 }
620
621 // parse the shortest paths json file
622 ifstream route_tables_file(ROUTE_FILE_PATH);
623 if (!route_tables_file.is_open()) {
624     cerr << "Failed to open routes file" << endl;
625     return 1;
626 }
627
628 // save the shortest paths
629 json route_tables, perf_indices;
630 route_tables_file >> route_tables;
631 route_tables_file.close();
632
633 ITER_COUNT = route_tables["iter"][day - 1];
634 perf_indices = route_tables["indices"];
635 route_tables = route_tables["routes"];
636
637 // create the layout graph for the school
638 Graph graph = createSchoolGraph("../assets/paths.txt");
639
640 // run the algorithm for a period
641 iterSingleDay(day, route_tables, graph, perf_indices);
642
643 return 0;
644 }
```

## optiway/src/app\_init.rs

```
1 pub fn setup_custom_fonts(ctx: &egui::Context) {
2     let mut fonts = egui::FontDefinitions::default();
3
4     fonts.font_data.insert(
5         "inter_font".to_owned(),
6         egui::FontData::from_static(include_bytes!("../assets/fonts/Inter-
7 micfong.ttf"))
8     );
9     fonts.font_data.insert(
10        "jetbrains_mono_font".to_owned(),
11        egui::FontData::from_static(include_bytes!("../assets/fonts/JetBrainsMono-
12 Regular.ttf"))
13    );
14    fonts.font_data.insert(
15        "source_han_sans_font".to_owned(),
16        egui::FontData::from_static(include_bytes!(
17            "../assets/fonts/SourceHanSansSC-Regular.otf"))
18    );
19    fonts.font_data.insert(
20        "material_design_icons_font".to_owned(),
21        egui::FontData::from_static(include_bytes!(
22            "../assets/fonts/MaterialDesignIcons.ttf"))
23    );
24    fonts.families
25        .entry(egui::FontFamily::Proportional)
26        .or_default()
27        .insert(0, "inter_font".to_owned());
28    fonts.families
29        .entry(egui::FontFamily::Proportional)
30        .or_default()
31        .insert(1, "source_han_sans_font".to_owned());
32    fonts.families
33        .entry(egui::FontFamily::Proportional)
34        .or_default()
35        .insert(2, "material_design_icons_font".to_owned());
36    fonts.families
37        .entry(egui::FontFamily::Proportional)
38        .or_default()
39        .insert(3, "material_symbols_font".to_owned());
40
41    fonts.families
42        .entry(egui::FontFamily::Monospace)
43        .or_default()
44        .insert(0, "jetbrains_mono_font".to_owned());
45
46    // Tell egui to use these fonts:
47    ctx.set_fonts(fonts);
48 }
49
50 pub fn setup_custom_styles(ctx: &egui::Context) {
51     let mut style: egui::Style = (*ctx.style()).clone();
52     style.visuals.window_rounding = (0.0).into();
```

```
53 |     style.visuals.menu_rounding = (0.0).into();
54 |     style.visuals.widgets.noninteractive.rounding = (0.0).into();
55 |     style.visuals.widgets.inactive.rounding = (0.0).into();
56 |     style.visuals.widgets.hovered.rounding = (0.0).into();
57 |     style.visuals.widgets.active.rounding = (0.0).into();
58 |     style.visuals.widgets.open.rounding = (0.0).into();
59 |     style.visuals.slider_trailing_fill = true;
60 |     style.visuals.override_text_color = Some(egui::Color32::from_rgb(180, 180, 180))
61 | );
62 | // style.animation_time = 1.0;
63 | ctx.set_style(style);
64 }
```

## optiway/src/app.rs

```
1 use std::{
2     collections::HashMap,
3     f32::consts::PI,
4     fmt::Display,
5     fs::{ self, File },
6     io::{ BufRead, BufReader, Read, Write },
7     path::{ Path, PathBuf },
8     process::{ Command, Stdio },
9     sync::{ Arc, Mutex },
10    thread,
11 };
12
13 use egui::{
14     color_picker,
15     emath,
16     pos2,
17     CentralPanel,
18     Color32,
19     ColorImage,
20     ComboBox,
21     Grid,
22     Layout,
23     ProgressBar,
24     Rect,
25     RichText,
26     Slider,
27     Stroke,
28     TextureHandle,
29     Window,
30 };
31 use num_format::{ Locale, ToFormattedString };
32 use rfd::FileDialog;
33
34 use crate::{
35     md_icons::material_design_icons, setup_custom_fonts,
36     setup_custom_styles
37 };
38 #[derive(Default, Clone, PartialEq, Eq)]
39 enum PathDisplay {
40     #[default]
41     Shortest,
42     Optimized,
43 }
44
45 impl Display for PathDisplay {
46     fn fmt(&self, f: &mut std::fmt::Formatter<'_>) → std::fmt::Result {
47         match self {
48             PathDisplay::Shortest => write!(f, "Shortest route"),
49             PathDisplay::Optimized => write!(f, "Optimized route"),
50         }
51     }
52 }
53 #[derive(Default, Clone, PartialEq, Eq)]
54 enum TimetableValidationStatus {
55     #[default]
56     Ready,
```

```

56     Validating(i32, String),
57     Failed(String),
58     Successful,
59 }
60
61 impl TimetableValidationStatus {
62     fn get_error_message(&self) → String {
63         match self {
64             TimetableValidationStatus::Failed(message) => message.clone(),
65             _ => String::new(),
66         }
67     }
68 }
69
70 #[derive(Default, Clone, PartialEq, Eq, serde::Deserialize, serde::Serialize)]
71 struct MainJSONOutputFile {
72     iter: [u64; 5],
73     indices: HashMap<u32, HashMap<usize, u128>>,
74     routes: HashMap<String, HashMap<u32, HashMap<usize, String>>>,
75 }
76
77 #[derive(Default, Clone, PartialEq, Eq, serde::Deserialize, serde::Serialize)]
78 struct SubJSONOutputFile {
79     iter: u64,
80     indices: HashMap<u32, HashMap<usize, u128>>,
81     routes: HashMap<String, HashMap<u32, HashMap<usize, String>>>,
82 }
83
84 #[derive(Default, Clone, PartialEq, Eq)]
85 enum OptimizationStatus {
86     #[default]
87     ParamInput,
88     Ready,
89     Calculating,
90     AbortSignal,
91     Failed(String),
92 }
93
94 #[derive(Default)]
95 struct TimetableFileInfo {
96     filename: String,
97     filepath: PathBuf,
98     student_count: Arc<Mutex<Option<i32>>>,
99     session_count: Arc<Mutex<Option<i32>>>,
100    validation_status: Arc<Mutex<TimetableValidationStatus>>,
101    timetable: Arc<Mutex<Option<serde_json::Value>>>,
102 }
103
104 #[derive(Default, Clone, PartialEq, Eq)]
105 enum PathGenerationStatus {
106     #[default]
107     Ready,
108     Generating(i32, String),
109     Failed(String),
110     LoadingJSON,
111     Successful,
112 }
113

```

```

114 impl PathGenerationStatus {
115     fn is_generating(&self) → bool {
116         matches!(self, PathGenerationStatus::Generating(_, _))
117     }
118
119     fn is_loading_json(&self) → bool {
120         matches!(self, PathGenerationStatus::LoadingJSON)
121     }
122 }
123
124 #[derive(Default, Clone, PartialEq, Eq)]
125 enum CongestionStatus {
126     #[default]
127     Ready,
128     Generating(i32, String),
129     Failed(String),
130     GeneratingPI(i32, String),
131     Successful,
132 }
133 impl CongestionStatus {
134     fn is_generating(&self) → bool {
135         if let CongestionStatus::Generating(_, _) = self {
136             true
137         } else {
138             matches!(self, CongestionStatus::GeneratingPI(_, _))
139         }
140     }
141 }
142
143 type Routes = HashMap<String, HashMap<u32, HashMap<usize, String>>>;
144 type CongestionPoint = HashMap<u32, HashMap<usize, HashMap<String, u32>>>;
145 type CongestionPath = HashMap<u32, HashMap<usize, HashMap<(String, String), u32>>>;
146
147 struct CongestionStatistics {
148     point_count: HashMap<u32, HashMap<usize, Vec<u32>>>,
149     path_count: HashMap<u32, HashMap<usize, Vec<u32>>>,
150 }
151
152 impl Default for CongestionStatistics {
153     fn default() → Self {
154         Self {
155             point_count: {
156                 let mut point_count = HashMap::new();
157                 for day in 1..=5 {
158                     point_count.insert(day, HashMap::new());
159                     for period in 0..=11 {
160                         point_count
161                             .get_mut(&day)
162                             .unwrap()
163                             .insert(period, vec![0; 7]);
164                     }
165                 }
166                 point_count
167             },
168             path_count: {
169                 let mut path_count = HashMap::new();
170                 for day in 1..=5 {
171                     path_count.insert(day, HashMap::new());
172                 }
173             }
174         }
175     }
176 }

```

```

172         for period in 0..=11 {
173             path_count
174                 .get_mut(&day)
175                 .unwrap()
176                 .insert(period, vec![0; 7]);
177             }
178         }
179     path_count
180 },
181 }
182 }
183 }
184
185 pub struct OptiWayApp {
186     selected_student: Option<String>,
187     student_list: Arc<Mutex<Vec<String>>>,
188     selected_period: usize,
189     selected_day: u32,
190     selected_floor: [bool; 9],
191     selected_floor_index: usize,
192     textures: Vec<Option<TextureHandle>>,
193     inactive_brightness: u8,
194     projection_coords: HashMap<String, [i32; 3]>,
195     active_path_color: Color32,
196     inactive_path_color: Color32,
197     show_path_window: bool,
198     show_json_validation: bool,
199     timetable_file_info: TimetableFileInfo,
200     student_number_search: String,
201     path_generation_status: Arc<Mutex<PathGenerationStatus>>,
202     show_path_gen_window: bool,
203     student_routes_shortest: Arc<Mutex<Option<Routes>>>,
204     student_routes_optimized: Arc<Mutex<Option<Routes>>>,
205     show_timetable_window: bool,
206     show_congestion_window: bool,
207     congestion_status: Arc<Mutex<CongestionStatus>>,
208     congestion_point_data: Arc<Mutex<CongestionPoint>>,
209     congestion_path_data: Arc<Mutex<CongestionPath>>,
210     congestion_point_data_opt: Arc<Mutex<CongestionPoint>>,
211     congestion_path_data_opt: Arc<Mutex<CongestionPath>>,
212     maximum_congestion: Arc<Mutex<u32>>,
213     maximum_congestion_opt: Arc<Mutex<u32>>,
214     congestion_statistics: Arc<Mutex<CongestionStatistics>>,
215     congestion_statistics_opt: Arc<Mutex<CongestionStatistics>>,
216     show_congestion: bool,
217     congestion_filter: u32,
218     show_congestion_path: bool,
219     show_congestion_point: bool,
220     show_pi_window: bool,
221     #[allow(unused)]
222     shortest_paths_json: HashMap<String, String>,
223     shortest_paths_content: Arc<Mutex<String>>,
224     show_pi_shortest: bool,
225     performance_indices_shortest: Arc<Mutex<HashMap<u32, HashMap<usize, u128>>>>,
226     performance_indices_optimized: Arc<Mutex<HashMap<u32, HashMap<usize, u128>>>>,
227     path_distances: Arc<Mutex<HashMap<String, HashMap<String, u32>>>>,
228     optimization_status: Arc<Mutex<OptimizationStatus>>,
229     show_optimization_window: bool,

```

```

230     param_batch_size: u32,
231     param_save_every: u32,
232     param_use_shortest_path: bool,
233     param_day: u32,
234     param_filename: String,
235     param_filepath: PathBuf,
236     current_iter: Arc<Mutex<[u64; 5]>>,
237     current_period_iter: Arc<Mutex<[u64; 5]>>,
238     path_display: PathDisplay,
239 }
240
241 impl Default for OptiWayApp {
242     fn default() → Self {
243         let mut floors = [false; 9];
244         floors[0] = true;
245         Self {
246             selected_student: Default::default(),
247             student_list: Default::default(),
248             selected_period: 0,
249             selected_day: 1,
250             selected_floor: floors,
251             selected_floor_index: 0,
252             textures: vec![None; 9],
253             inactive_brightness: 64,
254             projection_coords: serde_yaml
255                 ::from_str(include_str!("../assets/projection-coords-
flatten.yaml"))
256                 .unwrap(),
257             active_path_color: Color32::from_rgb(0xec, 0x6f, 0x27),
258             inactive_path_color: Color32::from_gray(0x61),
259             show_path_window: false,
260             show_json_validation: false,
261             timetable_file_info: Default::default(),
262             student_number_search: Default::default(),
263             path_generation_status: Default::default(),
264             show_path_gen_window: false,
265             student_routes_shortest: Default::default(),
266             student_routes_optimized: Default::default(),
267             show_timetable_window: false,
268             show_congestion_window: false,
269             congestion_status: Default::default(),
270             congestion_point_data: Arc::new(
271                 Mutex::new({
272                     let mut congestion_data = CongestionPoint::new();
273                     for day in 1..=5 {
274                         congestion_data.insert(day, HashMap::new());
275                         for period in 0..=11 {
276                             congestion_data.get_mut(&day).unwrap().insert(period,
277                                 HashMap::new());
278                         }
279                     }
280                 })
281             ),
282             congestion_path_data: Arc::new(
283                 Mutex::new({
284                     let mut congestion_data = CongestionPath::new();
285                     for day in 1..=5 {
286                         congestion_data.insert(day, HashMap::new());
287                     }
288                 })
289             )
290         }
291     }
292 }
```

```

287             for period in 0..=11 {
288                 congestion_data.get_mut(&day).unwrap().insert(period,
289             );
290         }
291     }
292 }
293 ),
294 congestion_point_data_opt: Arc::new(
295     Mutex::new({
296         let mut congestion_data = CongestionPoint::new();
297         for day in 1..=5 {
298             congestion_data.insert(day, HashMap::new());
299             for period in 0..=11 {
300                 congestion_data.get_mut(&day).unwrap().insert(period,
301             HashMap::new());
302             }
303         }
304     })
305 ),
306 congestion_path_data_opt: Arc::new(
307     Mutex::new({
308         let mut congestion_data = CongestionPath::new();
309         for day in 1..=5 {
310             congestion_data.insert(day, HashMap::new());
311             for period in 0..=11 {
312                 congestion_data.get_mut(&day).unwrap().insert(period,
313             HashMap::new());
314             }
315         }
316     })
317 ),
318 congestion_statistics: Default::default(),
319 congestion_statistics_opt: Default::default(),
320 maximum_congestion: Default::default(),
321 maximum_congestion_opt: Default::default(),
322 show_congestion: false,
323 congestion_filter: 0,
324 show_congestion_path: true,
325 show_congestion_point: true,
326 show_pi_window: false,
327 shortest_paths_json: serde_json
328     ::from_str(include_str!("../assets/shortest_paths.json"))
329     .unwrap(),
330 performance_indices_shortest: {
331     let mut performance_indices = HashMap::new();
332     for day in 1..=5 {
333         performance_indices.insert(day, HashMap::new());
334         for period in 0..=11 {
335             performance_indices.get_mut(&day).unwrap().insert(period,
336             0);
337         }
338     }
339     Arc::new(Mutex::new(performance_indices))
340 },
341 performance_indices_optimized: {
342     let mut performance_indices = HashMap::new();

```

```

342         for day in 1..=5 {
343             performance_indices.insert(day, HashMap::new());
344             for period in 0..=11 {
345                 performance_indices.get_mut(&day).unwrap().insert(period,
346                     0);
347             }
348         }
349     },
350     show_pi_shortest: true,
351     path_distances: Arc::new(
352         Mutex::new({
353             let mut path_distances = HashMap::new();
354             for line in include_str!("../assets/paths.txt").lines() {
355                 let line = line.to_owned();
356                 let mut line = line.split(' ');
357                 let room1 = line.next().unwrap().to_owned();
358                 let room2 = line.next().unwrap().to_owned();
359                 let distance = line.next().unwrap().parse::<u32>()
360                     .unwrap();
361                 // insert if not present
362                 if !path_distances.contains_key(&room1) {
363                     path_distances.insert(room1.clone(), HashMap::new());
364                 }
365                 path_distances.get_mut(&room1).unwrap()
366                     .insert(room2.clone(), distance);
367                 if !path_distances.contains_key(&room2) {
368                     path_distances.insert(room2.clone(), HashMap::new());
369                 }
370                 path_distances
371             }
372         },
373         optimization_status: Default::default(),
374         show_optimization_window: false,
375         param_batch_size: 100,
376         param_save_every: 2500,
377         param_use_shortest_path: true,
378         param_day: 1,
379         param_filename: Default::default(),
380         param_filepath: Default::default(),
381         shortest_paths_content: Default::default(),
382         current_iter: Default::default(),
383         current_period_iter: Default::default(),
384         path_display: Default::default(),
385     });
386 }
387 }
388
389 impl OptiWayApp {
390     pub fn new(cc: &eframe::CreationContext<'_>) → Self {
391         setup_custom_fonts(&cc.egui_ctx);
392         setup_custom_styles(&cc.egui_ctx);
393
394         // if let Some(storage) = cc.storage {
395         //     return eframe::get_value(storage, eframe::APP_KEY)
396         .unwrap_or_default();
397         // }

```

```

397     Default::default()
398 }
399
400
401 fn show_path_generation_window(
402     &mut self,
403     ctx: &egui::Context,
404     current_path_status: PathGenerationStatus
405 ) {
406     Window::new("Calculating Path").show(ctx, |ui| {
407         ui.heading("Metadata");
408         Grid::new("path_generation_grid")
409             .num_columns(2)
410             .show(ui, |ui| {
411                 ui.label("Algorithm");
412                 ui.label("Minimize distance");
413                 ui.end_row();
414
415                 ui.label("Student count");
416                 if
417                     let Some(student_count) =
418                         *self.timetable_file_info.student_count
419                             .lock()
420                             .unwrap()
421                         {
422                             ui.label(format!("{} ", student_count));
423                         } else {
424                             ui.label("-");
425                         }
426                         ui.end_row();
427
428                 ui.label("Session count");
429                 if
430                     let Some(session_count) =
431                         *self.timetable_file_info.session_count
432                             .lock()
433                             .unwrap()
434                         {
435                             ui.label(format!("{} ", session_count));
436                         } else {
437                             ui.label("-");
438                         }
439                         ui.end_row();
440                     });
441                     ui.separator();
442                     match current_path_status {
443                         PathGenerationStatus::Ready => {
444                             *self.path_generation_status.lock().unwrap() =
445                             PathGenerationStatus::Generating(
446                                 0,
447                                 "Calculating path".to_owned()
448                             );
449                             let filepath = self.timetable_file_info.filepath.clone();
450                             let path_generation_status_arc =
451                             self.path_generation_status.clone();
452                             let student_paths_arc = self.student_routes_shortest.clone();
453                             let shortest_paths_content_arc =
454                             self.shortest_paths_content.clone();
455                             thread::spawn(move || {

```

```

451 *shortest_paths_content_arc.lock().unwrap() =
452     run_floyd_algorithm_cpp(
453         filepath,
454         path_generation_status_arc,
455         student_paths_arc
456     );
457 });
458 // let shortest_paths_json = self.shortest_paths_json.clone()
459 ;
460 shortest_paths_json, path_generation_status_arc, student_paths_arc));
461 }
462 PathGenerationStatus::Generating(_progress, message) => {
463 ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
464 ui.label(RichText::new(material_design_icons::MDI_MAP_SEARCH).size(32.0));
465         ui.label(message);
466         ui.spinner();
467     });
468 }
469 PathGenerationStatus::Failed(message) => {
470 ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
471         ui.label(
472             RichText::new(material_design_icons::MDI_SIGN_DIRECTION_REMOVE)
473                 .size(32.0)
474                 .color(Color32::from_rgb(0xe4, 0x37, 0x48))
475             );
476         ui.label("Path calculation failed");
477         ui.label(message);
478         if ui.button("Close").clicked() {
479             self.show_path_gen_window = false;
480         }
481     });
482 }
483 PathGenerationStatus::LoadingJSON => {
484 ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
485         ui.label(
486             RichText::new(material_design_icons::MDI_MAP_CHECK)
487                 .size(32.0)
488                 .color(Color32::from_rgb(0x14, 0xae, 0x52))
489             );
490         ui.label("Loading paths");
491         ui.label(
492             "The paths have been successfully calculated. OptiWay
493 is now loading the paths."
494         );
495         ui.spinner();
496     });
497 }
498 PathGenerationStatus::Successful => {
499 ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
500         ui.label(
501             RichText::new(material_design_icons::MDI_MAP_CHECK)
502                 .size(32.0)
503                 .color(Color32::from_rgb(0x14, 0xae, 0x52))
504             );
505     });
506 }

```

```

502             ui.label("Path calculation successful");
503             ui.label(
504                 "The paths have been successfully calculated and
505                 loaded. You may now view or export them."
506             );
507             if ui.button("Close").clicked() {
508                 self.show_path_gen_window = false;
509             }
510         });
511     });
512 };
513 }
514
515 fn show_optimization_window(
516     &mut self,
517     ctx: &egui::Context,
518     current_optimization_status: OptimizationStatus
519 ) {
520     Window::new("Route Optimization").show(ctx, |ui| {
521         match current_optimization_status {
522             OptimizationStatus::ParamInput => {
523                 ui.heading("Parameters");
524                 ui.add(
525                     Slider::new(&mut self.param_batch_size, 10..=200)
526                         .step_by(10.0)
527                         .text("Batch size")
528                 );
529                 ui.add(
530                     Slider::new(&mut self.param_save_every, 100..=5000)
531                         .step_by(100.0)
532                         .text("Iterations per save")
533                 );
534                 // ComboBox::from_label("Day of Week")
535                 //     .selected_text(convert_day_of_week(self.param_day))
536                 //     .show_ui(ui, |ui| {
537                 //         for i in 1..=5 {
538                 //             ui.selectable_value(
539                 //                 &mut self.param_day,
540                 //                 i,
541                 //                 convert_day_of_week(i),
542                 //             );
543                 //         }
544                 //     });
545                 ui.horizontal(|ui| {
546                     ui.checkbox(&mut self.param_use_shortest_path, "Use
shortest routes");
547                     ui.add_enabled_ui(!self.param_use_shortest_path, |ui| {
548                         if ui.button("Select route file").clicked() {
549                             let file = FileDialog::new()
550                                 .add_filter("JSON", &["json"])
551                                 .pick_file();
552                         if let Some(file) = file {
553                             self.param_filename = file
554                                 .file_name()
555                                 .unwrap()
556                                 .to_str()
557                                 .unwrap()
558                                 .to_owned();

```

```

559             self.param_filepath = file;
560         }
561     }
562 });
563 if !self.param_use_shortest_path {
564     ui.label(
565         if self.param_filename.is_empty() {
566             "No file selected.".to_owned()
567         } else {
568             self.param_filename.clone()
569         }
570     );
571 }
572 });
573 ui.separator();
574
575 ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
576     if ui.button("Reset parameters").clicked() {
577         self.param_batch_size = 100;
578         self.param_save_every = 2500;
579         self.param_use_shortest_path = true;
580         self.param_day = 1;
581         self.param_filename = Default::default();
582         self.param_filepath = Default::default();
583     }
584     if ui.button("Start").clicked() {
585         self.show_pi_window = true;
586         self.show_pi_shortest = false;
587         *self.optimization_status.lock().unwrap() =
588             OptimizationStatus::Ready;
589     }
590 }
591 OptimizationStatus::Ready => {
592     *self.optimization_status.lock().unwrap() =
593         OptimizationStatus::Calculating;
594     let param_use_shortest_path = self.param_use_shortest_path;
595     let param_batch_size = self.param_batch_size;
596     let param_save_every = self.param_save_every;
597     for day in 1..=5 {
598         let param_day = day;
599         let mut param_filepath = self.param_filepath.clone();
600         let optimization_status_arc =
601             self.optimization_status.clone();
602         let shortest_paths_content_arc =
603             self.shortest_paths_content.clone();
604         let current_iter_arc = self.current_iter.clone();
605         let performance_indices_shortest_json = serde_json
606             ::to_string(&
607             *self.performance_indices_shortest.lock().unwrap()
608             .unwrap());
609         let performance_indices_optimized_arc =
610             self.performance_indices_optimized.clone();
611         let performance_indices_shortest_arc =
612             self.performance_indices_shortest.clone();
613         let current_period_iter_arc =
614             self.current_period_iter.clone();
615         if param_use_shortest_path {
616             self.param_filepath = fs::canonicalize("./bin/routes.json")
617             .unwrap();
618         }
619     }
620 }

```

```

611     self.param_filename = "routes.json".to_owned();
612     self.param_use_shortest_path = false;
613 } else {
614     let mut file = File::open(&param_filepath).unwrap();
615     let mut content = String::new();
616     file.read_to_string(&mut content).unwrap();
617     let json: MainJSONOutputFile = serde_json::from_str(&
618         content).unwrap();
619
620     for day in 1..=5 {
621         *self.current_iter
622             .lock()
623             .unwrap()
624             .get_mut((day - 1) as usize)
625             .unwrap() = *json.iter.get((day - 1) as
626             usize).unwrap();
627     }
628     *self.performance_indices_optimized.lock().unwrap() =
629         json.indices;
630 }
631 thread::spawn(move || {
632     if param_use_shortest_path {
633         let content =
634             "{\"iter\":[0,0,0,0,0],\"indices\":"
635             &performance_indices_shortest_json +
636             ",\"routes\":\"" +
637             &shortest_paths_content_arc.lock().unwrap()
638             "}";
639         *performance_indices_optimized_arc.lock()
640             .unwrap() =
641             performance_indices_shortest_arc.lock()
642             .unwrap().clone();
643         let mut file = File::create("./bin/routes.json")
644             .unwrap();
645         file.write_all(content.as_bytes()).unwrap();
646         param_filepath = fs::canonicalize("../../../bin/routes.json").unwrap();
647     }
648     let bin_dir = fs::canonicalize("./bin").unwrap();
649     if
650         let Ok(mut opt_command) = Command::new("./optimization.out")
651             .current_dir(bin_dir)
652             .stdout(Stdio::piped())
653             .stdin(Stdio::piped())
654             .args([
655                 "-f",
656                 param_filepath.to_str().unwrap(),
657                 "-b",
658                 &param_batch_size.to_string(),
659                 "-s",
660                 &(param_save_every / 5).to_string(),
661                 "-d",
662                 &param_day.to_string(),
663             ])
664             .spawn()
665     {
666         if let Some(stdout) = opt_command.stdout.take() {
667             let reader = BufReader::new(stdout);
668             reader
669         }
670     }
671 }

```

```

662     .lines()
663     .map_while(Result::ok)
664     .for_each(|line| {
665         let report: Vec<&str> = line.split('
').collect();
666
667         .unwrap())[0]
668
669         .unwrap();
670
671         OptimizationStatus::AbortSignal
672
673         optimization_algorithm"
674
675     {
676         opt_command
677             .kill()
678             .expect(
679                 "Failed to interrupt
680             );
681     }
682
683         .unwrap())[0]
684
685         *performance_indices_optimized_arc
686
687
688         .lock()
689         .unwrap()
690         .get_mut(&param_day)
691
692         .lock()
693         .unwrap()
694         .get_mut(
695             &
696             report[3]
697             .parse::<u64>()
698             .unwrap() as
699             )
700     )
701         .unwrap() = report[4].parse::<u128>().unwrap();
702
703     } else {
704         *optimization_status_arc.lock().unwrap() =
705             OptimizationStatus::Failed(
706                 "Optimization algorithm terminated
707                 unexpectedly".to_owned()
708             );
709     }
710 } else {
711     *optimization_status_arc.lock().unwrap() =
712         OptimizationStatus::Failed(
713             "Failed to start optimization algorithm"
714             .to_owned()
715         );

```

```

711             }
712         });
713     }
714 }
715 OptimizationStatus::Calculating => {
716
717     ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
718         ui.label(RichText::new(material_design_icons::MDI_COG)
719             .size(32.0));
720         ui.label("Optimizing paths");
721         ui.label(
722             format!(
723                 "Total Iteration: {}",
724                 self.current_iter
725                     .lock()
726                     .unwrap()
727                     .iter()
728                     .sum::<u64>()
729                     .to_formatted_string(&Locale::fr)
730             )
731         );
732         ui.spinner();
733         if ui.button("Pause").clicked() {
734             *self.optimization_status.lock().unwrap() =
735                 OptimizationStatus::AbortSignal;
736             // read from routes.json and load the json
737             let mut file = File::open("./bin/routes.json")
738                 .unwrap();
739
740             let mut content = String::new();
741             file.read_to_string(&mut content).unwrap();
742             let mut json: MainJSONOutputFile = serde_json
743                 ::from_str(&content)
744                 .unwrap();
745             let mut sub_jsons: [SubJSONOutputFile; 5] =
746                 Default::default();
747
748             for i in 1..=5 {
749                 let mut file = File::open(
750                     format!("./bin/routes.json_{}.json", i)
751                 ).unwrap();
752                 let mut content = String::new();
753                 file.read_to_string(&mut content).unwrap();
754                 sub_jsons[i - 1] = serde_json::from_str(&content)
755
756             .unwrap();
757             sub_jsons[
758                 i - 1
759             ].indices
760                 .get(&(i as u32))
761                 .unwrap()
762                 .clone();
763             for student_number in sub_jsons[i - 1]
764
765                 *json.routes
766                     .get_mut(student_number)
767                     .unwrap()
768                     .get_mut(&(i as u32))
769                     .unwrap() = sub_jsons[i - 1].routes
770                     .get(student_number)
771                     .unwrap()
772
773         }
774     }
775 }
```

```

764             .get(&(i as u32))
765             .unwrap()
766             .clone();
767         }
768     }
769     .unwrap();
770     let mut file = File::create("./bin/routes.json")
771         file.write_all(
772             serde_json::to_string(&json).unwrap().as_bytes()
773         ).unwrap();
774     *self.student_routes_optimized.lock().unwrap() =
775     Some(json.routes);
776     }
777 }
778 OptimizationStatus::AbortSignal => {
779     ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
780         ui.label(
781             RichText::new(material_design_icons::MDI_COG_PAUSE)
782                 .size(32.0)
783                 .color(Color32::from_rgb(0xff, 0xc1, 0x07))
784         );
785         ui.label("Optimization paused");
786         ui.label("You may resume optimization at any time.");
787         if ui.button("Resume").clicked() {
788             *self.optimization_status.lock().unwrap() =
789             OptimizationStatus::Ready;
790         }
791         if ui.button("Close").clicked() {
792             self.show_optimization_window = false;
793             *self.optimization_status.lock().unwrap() =
794             OptimizationStatus::ParamInput;
795         }
796     });
797     OptimizationStatus::Failed(message) => {
798         ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
799             ui.label(
800                 RichText::new(material_design_icons::MDI_COG_OFF)
801                     .size(32.0)
802                     .color(Color32::from_rgb(0xe4, 0x37, 0x48))
803             );
804             ui.label("Route optimization failed");
805             ui.label(message);
806             if ui.button("Close").clicked() {
807                 self.show_optimization_window = false;
808             }
809         });
810     });
811 };
812 }
813
814 fn show_congestion_window(
815     &mut self,
816     ctx: &egui::Context,
817     current_congestion_status: CongestionStatus
818 ) {

```



```

876             }
877         }
878         congestion_data
879     };
880     let congestion_point_data_arc =
881 self.congestion_point_data.clone();
882     let congestion_path_data_arc =
883 self.congestion_path_data.clone();
884     let congestion_point_data_opt_arc =
885 self.congestion_point_data_opt.clone();
886     let congestion_path_data_opt_arc =
887 self.congestion_path_data_opt.clone();
888     let congestion_status_arc = self.congestion_status.clone();
889     let max_congestion_arc = self.maximum_congestion.clone();
890     let max_congestion_opt_arc =
891 self.maximum_congestion_opt.clone();
892     let congestion_statistics_arc =
893 self.congestion_statistics.clone();
894     let congestion_statistics_opt_arc =
895 self.congestion_statistics_opt.clone();
896     let path_distances_arc = self.path_distances.clone();
897     let mut rooms: Vec<String> = self.projection_coords
898         .clone()
899         .keys()
900         .map(|k| k.to_owned())
901         .collect();
902     let performance_indices_shortest_arc =
903         self.performance_indices_shortest.clone();
904     rooms.push("G".to_owned());
905     let student_routes = self.student_routes_shortest.lock()
906         .unwrap().clone();
907     if student_routes.is_none() {
908         *congestion_status_arc.lock().unwrap() =
909             "No path data available".to_owned()
910     };
911     return;
912 }
913 let student_routes = student_routes.unwrap();
914 let student_routes_opt = self.student_routes_optimized.lock()
915     .unwrap().clone();
916 thread::spawn(move || {
917     if let Some(student_routes) = student_routes_opt {
918         for day in 1..=5 {
919             for period in 0..=11 {
920                 for room in &rooms {
921                     congestion_point_data_opt_arc
922                         .lock()
923                         .unwrap()
924                         .get_mut(&day)
925                         .unwrap()
926                         .get_mut(&period)
927                         .unwrap()
928                         .insert(room.to_owned(), 0);
929                 }
930             }
931         }
932     }
933     for student in &student_routes {
934         for day in 1..=5 {
935             for period in 0..=11 {
936                 let rooms = student_routes

```

```

927     .get(student.0)
928     .unwrap()
929     .get(&day)
930     .unwrap()
931     .get(&period)
932     .unwrap()
933     .split(' ')
934     .collect::<Vec<&str>>();
935     let mut previous_room = "";
936     for room in rooms {
937         if room.is_empty() || room == "G" {
938             continue;
939         }
940         if !previous_room.is_empty() {
941             let contains =
942                 .lock()
943                 .unwrap()
944                 .get(&day)
945                 .unwrap()
946                 .get(&period)
947                 .unwrap()
948                 .contains_key(
949                     &
950                     );
951             if contains {
952                 *congestion_path_data_opt_arc
953                     .lock()
954                     .unwrap()
955                     .get_mut(&day)
956                     .unwrap()
957                     .get_mut(&period)
958                     .unwrap()
959                     .get_mut(
960                         &
961                         previous_room.to_owned(),
962                         room.to_owned(),
963                         )
964                     )
965                     .unwrap() += 1;
966             } else {
967                 congestion_path_data_opt_arc
968                     .lock()
969                     .unwrap()
970                     .get_mut(&day)
971                     .unwrap()
972                     .get_mut(&period)
973                     .unwrap()
974                     .insert(
975                         (
976                             previous_room.to_owned(),
977                             room.to_owned(),
978                             ),
979                             1
980                         );
981             }

```

```

982     congestion_path_data_opt_arc
983
984
985
986
987
988
989
990     previous_room.to_owned())
991
992
993     if contains {
994         *congestion_path_data_opt_arc
995             .lock()
996             .unwrap()
997             .get(&day)
998             .unwrap()
999             .get(&period)
1000            .unwrap()
1001            .contains_key(
1002                &(room.to_owned(),
1003
1004                previous_room.to_owned(),
1005
1006
1007            } else {
1008                congestion_path_data_opt_arc
1009                    .lock()
1010                    .unwrap()
1011                    .get_mut(&day)
1012                    .unwrap()
1013                    .get_mut(&period)
1014                    .unwrap()
1015                    .insert(
1016                        (
1017                            room.to_owned(),
1018
1019                            previous_room.to_owned(),
1020
1021
1022            }
1023
1024     previous_room = room;
1025     *congestion_point_data_opt_arc
1026         .lock()
1027         .unwrap()
1028         .get_mut(&day)
1029         .unwrap()
1030         .get_mut(&period)
1031         .unwrap()
1032         .get_mut(room)
1033         .unwrap() += 1;
1034     let cur_max_congestion =
1035         *max_congestion_opt_arc
1036         .lock()
1037         .unwrap();

```

```

1037     .unwrap() =
1038             *max_congestion_opt_arc.lock()
1039             cur_max_congestion.max(
1040                 *congestion_point_data_opt_arc
1041                     .lock()
1042                     .unwrap()
1043                     .get_mut(&day)
1044                     .unwrap()
1045                     .get_mut(&period)
1046                     .unwrap()
1047                     .get_mut(room)
1048                     .unwrap()
1049             );
1050         }
1051     }
1052 }
1053 for day in 1..=5 {
1054     for period in 0..=11 {
1055         for room in &rooms {
1056             let room_congestion =
1057                 *congestion_point_data_opt_arc
1058                     .lock()
1059                     .unwrap()
1060                     .get_mut(&day)
1061                     .unwrap()
1062                     .get_mut(&period)
1063                     .unwrap()
1064                     .get_mut(room)
1065                     .unwrap();
1066             congestion_statistics_opt_arc
1067                     .lock()
1068                     .unwrap()
1069                     .point_count.get_mut(&day)
1070                     .unwrap()
1071                     .get_mut(&period)
1072                     .unwrap()
1073             [congestion_range_index(room_congestion)] += 1;
1074         }
1075     }
1076     for day in 1..=5 {
1077         for period in 0..=11 {
1078             congestion_path_data_opt_arc
1079                     .lock()
1080                     .unwrap()
1081                     .get_mut(&day)
1082                     .unwrap()
1083                     .get_mut(&period)
1084                     .unwrap()
1085                     .iter_mut()
1086                     .for_each(|(_, congestion)| {
1087                         congestion_statistics_opt_arc
1088                             .lock()
1089                             .unwrap()
1090                             .path_count.get_mut(&day)
1091                             .unwrap()
1092                             .get_mut(&period)
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2597
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2697
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2796
2797
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2897
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3197
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3297
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
33
```



```

1148     (previous_room.to_owned(), room.to_owned())
1149         )
1150             .unwrap() += 1;
1151     } else {
1152         congestion_path_data_arc
1153             .lock()
1154             .unwrap()
1155             .get_mut(&day)
1156             .unwrap()
1157             .get_mut(&period)
1158             .unwrap()
1159             .insert(
1160                 (previous_room.to_owned(),
1161                  , room.to_owned()),
1162                  1
1163              );
1164     let contains =
1165         .lock()
1166         .unwrap()
1167         .get(&day)
1168         .unwrap()
1169         .get(&period)
1170         .unwrap()
1171         .contains_key(
1172             &(room.to_owned(),
1173 previous_room.to_owned())
1174         );
1175     if contains {
1176         *congestion_path_data_arc
1177             .lock()
1178             .unwrap()
1179             .get_mut(&day)
1180             .unwrap()
1181             .get_mut(&period)
1182             .unwrap()
1183             .get_mut(
1184                 &(room.to_owned(),
1185 previous_room.to_owned())
1186             )
1187             .unwrap() += 1;
1188     } else {
1189         congestion_path_data_arc
1190             .lock()
1191             .unwrap()
1192             .get_mut(&day)
1193             .unwrap()
1194             .get_mut(&period)
1195             .unwrap()
1196             .insert(
1197                 (room.to_owned(),
1198                  , room.to_owned()),
1199                  1
1200              );
1201 }

```

```
1202
1203
1204
1205
1206
1207
1208
1209
1210     *max_congestion_arc
1211         .lock()
1212         .unwrap()
1213         .get_mut(&day)
1214         .unwrap()
1215         .get_mut(&period)
1216         .unwrap()
1217         .get_mut(room)
1218         .unwrap() += 1;
1219         let cur_max_congestion =
1220             *max_congestion_arc.lock().unwrap() =
1221                 cur_max_congestion.max(
1222                     *congestion_point_data_arc
1223                         .lock()
1224                         .unwrap()
1225                         .get_mut(&day)
1226                         .unwrap()
1227                         .get_mut(&period)
1228                         .unwrap()
1229                         .get_mut(room)
1230                         .unwrap()
1231                     );
1232             );
1233         }
1234     }
1235     }
1236     }
1237     }
1238     }
1239     }
1240     }
1241     }
1242     }
1243     }
1244     }
1245     }
1246     }
1247     }
1248     [congestion_range_index(room_congestion)] += 1;
1249     }
1250   }
1251   for day in 1..=5 {
1252     for period in 0..=11 {
1253       congestion_path_data_arc
1254         .lock()
1255         .unwrap()
1256         .get_mut(&day)
1257         .unwrap()
```

```

1258
1259
1260
1261     .get_mut(&period)
1262     .unwrap()
1263     .iter_mut()
1264     .for_each(|(_, congestion)| {
1265         congestion_statistics_arc
1266             .lock()
1267             .unwrap()
1268             .path_count.get_mut(&day)
1269             .unwrap()
1270             .get_mut(&period)
1271             .unwrap()
1272     })
1273     [congestion_range_index(*congestion)] += 1;
1274 }
1275 }
1276 *congestion_status_arc.lock().unwrap() =
1277 CongestionStatus::GeneratingPI(
1278     0,
1279     "Calculating performance indices".to_owned()
1280 );
1281 let mut performance_indices_shortest = HashMap::new();
1282 let path_distances = path_distances_arc.lock().unwrap()
1283 .clone();
1284 for day in 1..=5 {
1285     performance_indices_shortest.insert(day,
1286         HashMap::new());
1287
1288     for period in 0..=11 {
1289         let mut index = 0f64;
1290         for student in &student_routes {
1291             let rooms = student_routes
1292                 .get(student.0)
1293                 .unwrap()
1294                 .get(&day)
1295                 .unwrap()
1296                 .get(&period)
1297                 .unwrap()
1298                 .split(' ')
1299                 .collect::<Vec<&str>>();
1300             let mut previous_room = "";
1301             for room in rooms {
1302                 if room.is_empty() || room == "G" {
1303                     continue;
1304                 }
1305                 if !previous_room.is_empty() {
1306                     let path_distance = *path_distances
1307                         .get(previous_room)
1308                         .unwrap()
1309                         .get(room)
1310                         .unwrap_or_else(|| {
1311                             println!(
1312                                 "{:?}",
1313                                 previous_room,
1314                                 room
1315                             )
1316                         });
1317                     panic!("Path distance not found:
1318                           previous_room,
1319                           room
1320                         ")
1321                 }
1322             }
1323         }
1324     }
1325 }
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

1312                                     }) as f64;
1313         *congestion_path_data_arc
1314             .lock()
1315             .unwrap()
1316             .get(&day)
1317             .unwrap()
1318             .get(&period)
1319             .unwrap()
1320             .get(&(previous_room.to_owned(),
1321 room.to_owned())))
1322             .unwrap() as f64;
1323         index += path_distance *
1324             (2.0 + ((path_congestion - 300.0)
1325 / 200.0).tanh());
1326     }
1327     previous_room = room;
1328 }
1329 performance_indices_shortest
1330     .get_mut(&day)
1331     .unwrap()
1332     .insert(period, index as u128);
1333 }
1334 }
1335 // println!("{}",
1336 performance_indices_shortest).unwrap();
1337 *performance_indices_shortest_arc.lock().unwrap() =
1338     performance_indices_shortest;
1339 *congestion_status_arc.lock().unwrap() =
1340     CongestionStatus::Successful;
1341 }
1342 CongestionStatus::GeneratingPI(_, message) => {
1343     ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
1344         ui.label(RichText::new(material_design_icons::MDI_TIMER)
1345             .size(32.0));
1346         ui.label(message);
1347         ui.spinner();
1348     });
1349 CongestionStatus::Generating(_, message) => {
1350     ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
1351         ui.label(
1352             RichText::new(material_design_icons::MDI_TRAFFIC_LIGHT).size(32.0)
1353             );
1354         ui.label(message);
1355         ui.spinner();
1356     });
1357 CongestionStatus::Failed(message) => {
1358     ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
1359         ui.label(
1360             RichText::new(material_design_icons::MDI_TRAFFIC_LIGHT)
1361             .size(32.0)

```

```

1362             .color(Color32::from_rgb(0xe4, 0x37, 0x48))
1363         );
1364         ui.label("Congestion evaluation failed");
1365         ui.label(message);
1366         if ui.button("Close").clicked() {
1367             self.show_congestion_window = false;
1368         }
1369     });
1370 }
1371 CongestionStatus::Successful => {
1372     ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
1373         ui.label(
1374             RichText::new(material_design_icons::MDI_TRAFFIC_LIGHT)
1375                 .size(32.0)
1376                 .color(Color32::from_rgb(0x14, 0xae, 0x52))
1377             );
1378         ui.label("Congestion evaluation successful");
1379         ui.label(
1380             "The congestion has been successfully evaluated and
1381             loaded. You may now view them on the projection."
1382         );
1383         ui.label(
1384             format!(
1385                 "Maximum congestion: {}",
1386                 self.maximum_congestion.lock().unwrap()
1387             )
1388         );
1389         if ui.button("Close").clicked() {
1390             self.show_congestion_window = false;
1391         }
1392     });
1393 };
1394 });
1395 }
1396
1397 fn show_json_validation_window(
1398     &mut self,
1399     ctx: &egui::Context,
1400     current_validation_status: TimetableValidationStatus
1401 ) {
1402     Window::new("Timetable Validation").show(ctx, |ui| {
1403         ui.heading("Metadata");
1404         Grid::new("timetable_validation_grid")
1405             .num_columns(2)
1406             .show(ui, |ui| {
1407                 ui.label("Filename");
1408                 ui.label(self.timetable_file_info.filename.to_string());
1409                 ui.end_row();
1410
1411                 ui.label("Student count");
1412                 if
1413                     let Some(student_count) =
1414                         *self.timetable_file_info.student_count
1415                             .lock()
1416                             .unwrap()
1417                         {

```

```

1417             ui.label(format!("{}", student_count));
1418         } else {
1419             ui.label("-");
1420         }
1421         ui.end_row();
1422
1423         ui.label("Session count");
1424         if
1425             let Some(session_count) =
1426             *self.timetable_file_info.session_count
1427                 .lock()
1428                 .unwrap()
1429             {
1430                 ui.label(format!("{}", session_count));
1431             } else {
1432                 ui.label("-");
1433             }
1434             ui.end_row();
1435         );
1436         ui.separator();
1437         match current_validation_status {
1438             TimetableValidationStatus::Ready => {
1439                 *self.timetable_file_info.validation_status.clone().lock()
1440                 .unwrap() =
1441                     TimetableValidationStatus::Validating(0, "Ready to
1442                     validate".to_owned());
1443                     let filepath = self.timetable_file_info.filepath.clone();
1444                     let projection_coords = self.projection_coords.clone();
1445                     let validation_status_arc =
1446                     self.timetable_file_info.validation_status.clone();
1447                     let student_count_arc =
1448                     self.timetable_file_info.student_count.clone();
1449                     let session_count_arc =
1450                     self.timetable_file_info.session_count.clone();
1451                     let timetable_arc =
1452                     self.timetable_file_info.timetable.clone();
1453                     let student_list_arc = self.student_list.clone();
1454                     thread::spawn(move || {
1455                         let mut rooms: Vec<String> = projection_coords
1456                             .keys()
1457                             .map(|k| k.to_owned())
1458                             .collect();
1459                         *validation_status_arc.lock().unwrap() =
1460                             TimetableValidationStatus::Validating(
1461                                 0,
1462                                 "Validating student numbers...".to_owned()
1463                             );
1464                         rooms.push("G".into());
1465                         let mut timetable_file = std::fs::File::open(filepath)
1466                             .unwrap();
1467                         let mut timetable_json = String::new();
1468                         if timetable_file.read_to_string(&mut timetable_json)
1469                             .is_err() {
1470                                 *validation_status_arc.lock().unwrap() =
1471                                     TimetableValidationStatus::Failed(
1472                                         "Failed to read timetable file".to_owned()
1473                                     );
1474                                     return;
1475                             }
1476                         let timetable: serde_json::Value = match
1477                             serde_json::from_str(&timetable_json)

```

```

1469
1470         Ok(t) => t,
1471         Err(e) => {
1472             *validation_status_arc.lock().unwrap() =
1473                 TimetableValidationStatus::Failed(
1474                     format!("Invalid JSON format in timetable
1475                         file: {}", e)
1476                     );
1477                     return;
1478                 }
1479             let mut student_count = 0;
1480
1481             if timetable.as_object().is_none() {
1482                 *validation_status_arc.lock().unwrap() =
1483                     TimetableValidationStatus::Failed(
1484                         "Invalid timetable file format: the JSON file
1485                         is not a map".to_owned()
1486                     );
1487                     return;
1488                 }
1489             for student_key in timetable.as_object().unwrap().keys()
1490             {
1491                 if
1492                     student_key.chars().all(char::is_numeric) &&
1493                     4 <= student_key.len() &&
1494                     student_key.len() <= 5
1495                 {
1496                     student_count += 1;
1497                     *student_count_arc.lock().unwrap() =
1498                         Some(student_count);
1499                 } else {
1500                     *validation_status_arc.lock().unwrap() =
1501                         TimetableValidationStatus::Failed(
1502                             format!("Invalid student number: \"{}\"", student_key)
1503                         );
1504                     return;
1505                 }
1506
1507             *validation_status_arc.lock().unwrap() =
1508                 TimetableValidationStatus::Validating(
1509                     10,
1510                     "Validating days of the week...".to_owned()
1511                 );
1512             for (student_key, week_timetable) in
1513                 timetable.as_object().unwrap() {
1514                 if week_timetable.as_object().is_none() {
1515                     *validation_status_arc.lock().unwrap() =
1516                         TimetableValidationStatus::Failed(
1517                             format!("Invalid timetable file format:
1518                         student {}'s timetable is not a map", student_key)
1519                             );
1520                     return;
1521                 }
1522                 let mut days_of_week = [false; 5];
1523                 for day_key in week_timetable.as_object().unwrap()
1524                     .keys() {
1525                         if

```

```

1521             day_key.chars().all(char::is_numeric) &&
1522             day_key.parse::<i32>().unwrap() <= 5 &&
1523             day_key.parse::<i32>().unwrap() >= 1
1524         {
1525             days_of_week[
1526                 (day_key.parse::<i32>().unwrap() as
1527                  usize) - 1
1528             ] = true;
1529         } else {
1530             *validation_status_arc.lock().unwrap() =
1531                 TimetableValidationStatus::Failed(
1532                     format!(
1533                         "Student {} has an invalid day of
1534                         week: \"{}\"",
1535                         student_key,
1536                         day_key
1537                     )
1538                 );
1539             return;
1540         }
1541     }
1542     if days_of_week.contains(&false) {
1543         *validation_status_arc.lock().unwrap() =
1544             TimetableValidationStatus::Failed(
1545                 format!(
1546                     "Student {} has an incomplete
1547                     timetable: missing day {}",
1548                     student_key,
1549                     days_of_week
1550                         .iter()
1551                         .enumerate()
1552                         .filter(|(_, &b)| !b)
1553                         .map(|(i, _)| i + 1)
1554                         .collect::<Vec<usize>>()
1555                         .iter()
1556                         .map(|i| i.to_string())
1557                         .collect::<Vec<String>>()
1558                         .join(", ")
1559                     )
1560                 );
1561             return;
1562         }
1563     }
1564     *validation_status_arc.lock().unwrap() =
1565         TimetableValidationStatus::Validating(
1566             20,
1567             "Validating periods...".to_owned()
1568         );
1569     for (student_key, week_timetable) in
1570         timetable.as_object().unwrap() {
1571         for (day_key, day_timetable) in
1572             week_timetable.as_object().unwrap() {
1573                 if day_timetable.as_object().is_none() {
1574                     *validation_status_arc.lock().unwrap() =
1575                         TimetableValidationStatus::Failed(
1576                             format!(
1577                                 "Invalid timetable file format:
1578                                 student {}'s timetable on day {} is not a map",
1579                                 student_key,

```

```

1575                                     day_key
1576                                         )
1577                                         );
1578                                         return;
1579 }
1580 let mut periods = [false; 10];
1581 for period_key in day_timetable.as_object()
1582 {
1583     if
1584         period_key.chars().all(char::is_numeric)
1585         &&
1586         period_key.parse::<i32>().unwrap() <= 10
1587         &&
1588         period_key.parse::<i32>().unwrap() >= 1
1589     {
1590         periods[
1591             (period_key.parse::<i32>().unwrap()
1592             as usize) - 1
1593         ] = true;
1594     } else {
1595         *validation_status_arc.lock().unwrap() =
1596             TimetableValidationStatus::Failed(
1597                 format!(
1598                     "Student {} has an invalid
1599                         student_key,
1600                         day_key,
1601                         period_key
1602                     )
1603                 );
1604             return;
1605     }
1606 }
1607 if periods.contains(&false) {
1608     *validation_status_arc.lock().unwrap() =
1609         TimetableValidationStatus::Failed(
1610             format!(
1611                 "Student {} has an incomplete
1612                     timetable on day {}: missing periods {}",
1613                     student_key,
1614                     day_key,
1615                     periods
1616                     .iter()
1617                     .enumerate()
1618                     .filter(|(_, &b)| !b)
1619                     .map(|(i, _)| i + 1)
1620                     .collect::<Vec<usize>>()
1621                     .iter()
1622                     .map(|i| i.to_string())
1623                     .collect::<Vec<String>>()
1624                     .join(", ")
1625                 )
1626             );
1627         return;
1628     }
1629 }
1630
1631 *validation_status_arc.lock().unwrap() =
1632     TimetableValidationStatus::Validating

```

```

1629          20,
1630          "Validating classrooms...".to_owned()
1631      );
1632      let mut sessions = 0;
1633      for (student_key, week_timetable) in
1634      timetable.as_object().unwrap() {
1635          for (day_key, day_timetable) in
1636          week_timetable.as_object().unwrap() {
1637              for (period_key, room) in
1638              day_timetable.as_object().unwrap() {
1639                  if !rooms.contains(&room.as_str().unwrap())
1640                  .to_owned()) {
1641                      *validation_status_arc.lock().unwrap() =
1642                      TimetableValidationStatus::Failed(
1643                          format!(
1644                              "Student {} has an invalid
1645                              classroom on day {} period {}: {}",
1646                              student_key,
1647                              day_key,
1648                              period_key,
1649                              room
1650                          )
1651                      );
1652                  }
1653                  sessions += 1;
1654                  if sessions % 1000 == 0 {
1655                      *session_count_arc.lock().unwrap() =
1656                      Some(sessions);
1657                      *validation_status_arc.lock().unwrap() =
1658                      TimetableValidationStatus::Validating(
1659                          20 +
1660                          (
1661                              (((sessions as f32) /
1662                              ((student_count * 10
1663                              * 5) as f32)) *
1664                              80.0) as i32
1665                          ),
1666                          "Validating classrooms..."
1667                      .to_owned()
1668                      );
1669                  }
1670                  }
1671                  }
1672                  }
1673                  }
1674                  }
1675                  *validation_status_arc.lock().unwrap() =
1676                  TimetableValidationStatus::Successful;
1677              });
1678          }
1679      TimetableValidationStatus::Validating(progress, message) => {

```

```

1680     ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
1681         ui.label(
1682             RichText::new(material_design_icons::MDI_CALENDAR_SEARCH).size(32.0)
1683                     );
1684             ui.label(message);
1685             ui.add(ProgressBar::new((progress as f32) / 100.0));
1686         });
1687     }
1688     TimetableValidationStatus::Failed(_) => {
1689         ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
1690             ui.label(
1691                 RichText::new(material_design_icons::MDI_CALENDAR_REMOVE)
1692                         .size(32.0)
1693                         .color(Color32::from_rgb(0xe4, 0x37, 0x48))
1694                     );
1695             ui.label("Validation failed");
1696             ui.label(current_validation_status.get_error_message());
1697             if ui.button("Close").clicked() {
1698                 self.show_json_validation = false;
1699             }
1700         });
1701     }
1702     TimetableValidationStatus::Successful => {
1703         ui.with_layout(Layout::top_down_justified(egui::Align::Center), |ui| {
1704             ui.label(
1705                 RichText::new(material_design_icons::MDI_CALENDAR_CHECK)
1706                         .size(32.0)
1707                         .color(Color32::from_rgb(0x14, 0xae, 0x52))
1708                     );
1709             ui.label("Validation successful");
1710             ui.label(
1711                 "The timetable has been imported successfully. You
may now proceed to the next step."
1712             );
1713             if ui.button("Close").clicked() {
1714                 self.show_json_validation = false;
1715             }
1716         });
1717     }
1718 }
1719 });
1720 }
1721
1722 fn show_timetable_window(&mut self, ctx: &egui::Context) {
1723     Window::new("Timetable")
1724         .open(&mut self.show_timetable_window)
1725         .show(ctx, |ui| {
1726             if
1727                 self.timetable_file_info.validation_status.lock().unwrap()
1728             .clone() ==
1729                 TimetableValidationStatus::Successful
1730             {
1731                 if let Some(student) = &self.selected_student {
1732                     ui.label(format!("{}'s Timetable", student));

```

```
1732             egui::Grid
1733                 ::new("timetable_grid")
1734                     .striped(true)
1735                     .show(ui, |ui| {
1736                         ui.label("Period");
1737                         ui.label("Monday");
1738                         ui.label("Tuesday");
1739                         ui.label("Wednesday");
1740                         ui.label("Thursday");
1741                         ui.label("Friday");
1742                         ui.end_row();
1743
1744             let timetable =
1745 self.timetable_file_info.timetable.lock().unwrap();
1746
1747         for period in 1..=10 {
1748             ui.label(format!("Period {}", period));
1749             for weekday in 1..=5 {
1750                 let mut session = timetable
1751                     .as_ref()
1752                     .unwrap()
1753                     .get(student)
1754                     .unwrap()
1755                     .get(weekday.to_string())
1756                     .unwrap()
1757                     .get(period.to_string())
1758                     .unwrap()
1759                     .as_str()
1760                     .unwrap();
1761                 if session == "G" {
1762                     session = " ";
1763                 }
1764                 ui.label(session);
1765             }
1766             ui.end_row();
1767         });
1768     } else {
1769         ui.label("No student selected.");
1770     }
1771 } else {
1772     ui.label("Timetable not imported.");
1773 }
1774 });
1775 }
1776
1777 fn show_pi_window(&mut self, ctx: &egui::Context) {
1778     egui::Window
1779         ::new("Performance Indices")
1780         .open(&mut self.show_pi_window)
1781         .show(ctx, |ui| {
1782             let performance_indices_shortest =
1783 self.performance_indices_shortest
1784                 .lock()
1785                 .unwrap();
1786             let performance_indices_optimized =
1787 self.performance_indices_optimized
1788                 .lock()
1789                 .unwrap();
```

```

1788         ui.horizontal(|ui| {
1789             .clicked() {
1790                 if ui.selectable_label(self.show_pi_shortest, "Shortest")
1791                     self.show_pi_shortest = true;
1792             }
1793
1794             .clicked() {
1795                 self.show_pi_shortest = false;
1796             }
1797             ui.separator();
1798             ui.label("Performance Indices Overview");
1799             let pi_matrix = if self.show_pi_shortest {
1800                 performance_indices_shortest
1801             } else {
1802                 performance_indices_optimized
1803             };
1804             let highlights = *self.current_period_iter.lock().unwrap();
1805             let optimization_status = self.optimization_status.lock()
1806             .unwrap().clone();
1807             Grid::new("shortest_pi_grid")
1808                 .striped(true)
1809                 .num_columns(6)
1810                 .show(ui, |ui| {
1811                     ui.label("");
1812                     ui.label("Monday");
1813                     ui.label("Tuesday");
1814                     ui.label("Wednesday");
1815                     ui.label("Thursday");
1816                     ui.label("Friday");
1817                     ui.end_row();
1818                     for index in 0..=11 {
1819                         ui.label(convert_periods(index));
1820                         for day in 1..=5 {
1821                             if
1822                                 (highlights[(day - 1) as usize] as usize) ==
1823                                 OptimizationStatus::Calculating
1824                                     {
1825                                         ui.label(
1826                                             RichText::new(
1827                                                 pi_matrix[&day][&index]
1828                                                 .to_formatted_string(&Locale::fr)
1829                                                 .color(Color32::from_rgb(0xec, 0x6f,
1830 0x27)))
1831                                         );
1832                                     } else {
1833                                         ui.label(
1834                                             pi_matrix[&day][&index]
1835                                             .to_formatted_string(&Locale::fr)
1836                                             );
1837                                         }
1838                                         ui.end_row();
1839                                     });
1840             ui.separator();
1841             ui.label("Current Period PI");

```

```

1840         ui.heading(
1841             pi_matrix[&self.selected_day][&self.selected_period]
1842             .to_formatted_string(
1843                 &Locale::fr
1844             )
1845             );
1846             ui.separator();
1847             ui.label("Current Day PI");
1848             ui.heading(
1849                 pi_matrix[&self.selected_day]
1850                     .values()
1851                     .sum::<u128>()
1852                     .to_formatted_string(&Locale::fr)
1853             );
1854             ui.separator();
1855             ui.label("Total PI");
1856             ui.heading(
1857                 pi_matrix
1858                     .values()
1859                     .map(|day| day.values().sum::<u128>())
1860                     .sum::<u128>()
1861                     .to_formatted_string(&Locale::fr)
1862             );
1863             ui.separator();
1864             ui.label(
1865                 "Performance indices measure the overall performance of the
1866                 routes. The lower the value, the better the performance."
1867             );
1868         });
1869     }
1870 #[allow(dead_code)]
1871 fn run_floyd_algorithm_cpp(
1872     filepath: PathBuf,
1873     path_generation_status_arc: Arc<Mutex<PathGenerationStatus>>,
1874     student_paths_arc: Arc<Mutex<Option<Routes>>>
1875 ) -> String {
1876     let bin_dir = fs::canonicalize("./bin").unwrap();
1877     if let Ok(binding) = fs::canonicalize(filepath) {
1878         let json_path = binding.to_str().unwrap();
1879         println!("{}\\n{}", bin_dir, json_path);
1880         if
1881             let Ok(mut floyd_command) = Command::new("./floyd.out")
1882                 .current_dir(&bin_dir)
1883                 .stdin(Stdio::piped())
1884                 .spawn()
1885             {
1886                 floyd_command.stdin.as_mut().unwrap().write_all(json_path.as_bytes())
1887             .unwrap();
1888                 let output = floyd_command.wait_with_output();
1889                 if output.is_err() {
1890                     *path_generation_status_arc.lock().unwrap() =
1891                         PathGenerationStatus::Failed(
1892                             "Failed to execute algorithm binary [floyd.out]".to_owned()
1893                         );
1894                 } else if output.as_ref().unwrap().status.success() {
1895                     *path_generation_status_arc.lock().unwrap() =
1896                         PathGenerationStatus::LoadingJSON;
1897                     let result_path = bin_dir.join("routes.json");

```

```

1895             let Ok(file_content) = fs::read_to_string(result_path) else {
1896                 *path_generation_status_arc.lock().unwrap() =
1897                     PathGenerationStatus::Failed(
1898                         "Failed to read result file [routes.json].".to_owned()
1899                     );
1900                     return "".to_owned();
1901                 };
1902                 let Ok(routes) = serde_json::from_str::<Routes>(&file_content)
1903             else {
1904                 *path_generation_status_arc.lock().unwrap() =
1905                     PathGenerationStatus::Failed(
1906                         "Failed to parse result file [routes.json].".to_owned()
1907                     );
1908                     return "".to_owned();
1909                 };
1910                 *student_paths_arc.lock().unwrap() = Some(routes);
1911                 *path_generation_status_arc.lock().unwrap() =
1912                     PathGenerationStatus::Successful;
1913                     return file_content;
1914                 } else {
1915                     *path_generation_status_arc.lock().unwrap() =
1916                         PathGenerationStatus::Failed(
1917                             format!(
1918                                 "Algorithm binary [floyd.out] exited with code {}.",
1919                                 output.unwrap().status.code().unwrap()
1920                             )
1921                         );
1922                     }
1923                 } else {
1924                     *path_generation_status_arc.lock().unwrap() =
1925                         PathGenerationStatus::Failed(
1926                             "Failed to execute algorithm binary [floyd.out].".to_owned()
1927                         );
1928                     }
1929                 }
1930
1931 #[allow(dead_code)]
1932 fn run_floyd_algorithm_rust(
1933     filepath: PathBuf,
1934     shortest_paths_json: HashMap<String, String>,
1935     path_generation_status_arc: Arc<Mutex<PathGenerationStatus>>,
1936     student_paths_arc: Arc<Mutex<Option<Routes>>>
1937 ) {
1938     type Timetable = HashMap<String, HashMap<u32, HashMap<usize, String>>>;
1939     let mut result: Routes = HashMap::new();
1940     let Ok(timetable_file) = File::open(filepath) else {
1941         *path_generation_status_arc.lock().unwrap() =
1942             PathGenerationStatus::Failed(
1943                 "Failed to find timetable file.".to_owned()
1944             );
1945             return;
1946         };
1947         let Ok(timetable_json) = serde_json::from_reader::<_, Timetable>
1948 (timetable_file) else {

```

```

1947     *path_generation_status_arc.lock().unwrap() =
1948     PathGenerationStatus::Failed(
1949         "Failed to parse timetable file. Please make sure the file is in JSON
1950         format.".to_owned()
1951     );
1952     return;
1953 };
1954     for student_number in timetable_json.keys() {
1955         let mut student_hashmap: HashMap<u32, HashMap<usize, String>> =
1956         HashMap::new();
1957         let mut student_timetable = timetable_json.get(student_number).unwrap()
1958             .clone();
1959             for day in 1..=5 {
1960                 let mut day_hashmap: HashMap<usize, String> = HashMap::new();
1961                 student_timetable.entry(day).or_insert_with(HashMap::new());
1962                 day_hashmap.insert(0, {
1963                     let first_room = student_timetable.get(&day).unwrap().get(&1)
1964                         .unwrap().to_owned();
1965                         if first_room == "G" {
1966                             "".to_owned()
1967                         } else {
1968                             shortest_paths_json.get(format!("G{}", first_room).as_str())
1969                         .unwrap().to_owned()
1970                         }
1971                     });
1972                     for period in 1..=5 {
1973                         day_hashmap.insert(period, {
1974                             let room_a = student_timetable
1975                                 .get(&day)
1976                                 .unwrap()
1977                                 .get(&period)
1978                                 .unwrap()
1979                                 .to_owned();
1980                             let room_b = student_timetable
1981                                 .get(&day)
1982                                 .unwrap()
1983                                 .get(&(period + 1))
1984                                 .unwrap()
1985                                 .to_owned();
1986                             if room_a == room_b {
1987                                 "".to_owned()
1988                             } else {
1989                                 shortest_paths_json
1990                                     .get(format!("{}{}", room_a, room_b).as_str())
1991                                     .unwrap()
1992                                     .to_owned()
1993                             }
1994                             day_hashmap.insert(6, {
1995                                 let first_room = student_timetable.get(&day).unwrap().get(&6)
1996                         .unwrap().to_owned();
1997                         if first_room == "G" {
1998                             "".to_owned()
1999                         } else {
2000                             shortest_paths_json.get(format!("{}G", first_room).as_str())
2001                             .unwrap().to_owned()
2002                         }
2003                     });
2004                     day_hashmap.insert(7, {
2005

```

```

1999     let first_room = student_timetable.get(&day).unwrap().get(&7)
2000         .unwrap().to_owned();
2001             if first_room == "G" {
2002                 "".to_owned()
2003             } else {
2004                 shortest_paths_json.get(format!("G{}", first_room).as_str())
2005             .unwrap().to_owned()
2006         }
2007     );
2008     for period in 7..=9 {
2009         day_hashmap.insert(period + 1, {
2010             let room_a = student_timetable
2011                 .get(&day)
2012                     .unwrap()
2013                         .get(&period)
2014                             .unwrap()
2015                               .to_owned();
2016             let room_b = student_timetable
2017                 .get(&day)
2018                     .unwrap()
2019                         .get(&(period + 1))
2020                             .unwrap()
2021                               .to_owned();
2022             if room_a == room_b {
2023                 "".to_owned()
2024             } else {
2025                 shortest_paths_json
2026                     .get(format!("{}{}", room_a, room_b).as_str())
2027                         .unwrap()
2028                           .to_owned()
2029             }
2030         );
2031         day_hashmap.insert(11, {
2032             let first_room = student_timetable.get(&day).unwrap().get(&10)
2033         .unwrap().to_owned();
2034             if first_room == "G" {
2035                 "".to_owned()
2036             } else {
2037                 shortest_paths_json.get(format!("{}G", first_room).as_str())
2038             .unwrap().to_owned()
2039             }
2040         );
2041         student_hashmap.insert(day, day_hashmap);
2042     }
2043     *student_paths_arc.lock().unwrap() = Some(result);
2044     *path_generation_status_arc.lock().unwrap() =
2045         PathGenerationStatus::Successful;
2046 }
2047 impl eframe::App for OptiWayApp {
2048     // fn save(&mut self, storage: &mut dyn eframe::Storage) {
2049     //     eframe::set_value(storage, eframe::APP_KEY, self);
2050     // }
2051     fn update(&mut self, ctx: &egui::Context, _frame: &mut eframe::Frame) {
2052         let current_validation_status =
self.timetable_file_info.validation_status

```

```

2053     .lock()
2054     .unwrap()
2055     .clone();
2056     let current_congestion_status = self.congestion_status.lock().unwrap()
2057     .clone();
2058     let current_path_status = self.path_generation_status.lock().unwrap()
2059     .clone();
2060     let current_optimization_status = self.optimization_status.lock()
2061     .unwrap().clone();
2062     egui::TopBottomPanel::top("top_panel").show(ctx, |ui| {
2063         egui::menu::bar(ui, |ui| {
2064             ui.label("OptiWay");
2065             egui::warn_if_debug_build(ui);
2066             ui.separator();
2067             if current_validation_status !=
2068                 TimetableValidationStatus::Successful {
2069                 ui.label(material_design_icons::MDI_CALENDAR_ALERT)
2070                 .on_hover_text(
2071                     "Timetable not imported."
2072                 );
2073                 ui.separator();
2074             }
2075             if self.selected_student.is_none() {
2076                 ui.label(material_design_icons::MDI_ACCOUNT_ALERT)
2077                 .on_hover_text(
2078                     "No student selected."
2079                 );
2080                 ui.separator();
2081             }
2082             if self.student_routes_shortest.lock().unwrap().is_none() {
2083                 ui.label(material_design_icons::MDI_SIGN_DIRECTION_REMOVE)
2084                 .on_hover_text(
2085                     "Path not calculated."
2086                 );
2087                 ui.separator();
2088             }
2089             ui.with_layout(Layout::right_to_left(egui::Align::Center), |ui| {
2090                 if self.show_json_validation {
2091                     ui.label("Validating timetable file");
2092                 } else if current_path_status.is_generating() {
2093                     ui.label("Calculating path");
2094                 } else if current_path_status.is_loading_json() {
2095                     ui.label("Loading paths");
2096                 } else if current_congestion_status.is_generating() {
2097                     ui.label("Evaluating congestion");
2098                 } else if current_optimization_status ==
2099                     OptimizationStatus::Calculating {
2100                     ui.label("Optimizing routes");
2101                 } else {
2102                     ui.label("Ready");
2103                 }
2104             });
}

```

```
2105
2106     egui::SidePanel::right("side_panel").show(ctx, |ui| {
2107         egui::ScrollArea::vertical().show(ui, |ui| {
2108             ui.with_layout(Layout::top_down_justified(egui::Align::LEFT),
2109             |ui| {
2110                 ui.heading("Command center").on_hover_text(
2111                     "Follow the order of the buttons from top to bottom to
2112                     complete the process.");
2113             });
2114             if ui.button("Import timetable").clicked() {
2115                 let file = FileDialog::new().add_filter("JSON", &["json"])
2116                 .pick_file();
2117                 if let Some(file) = file {
2118                     self.timetable_file_info.filename = file
2119                         .file_name()
2120                         .unwrap()
2121                         .to_str()
2122                         .unwrap()
2123                         .to_owned();
2124                     self.timetable_file_info.filepath = file;
2125                     self.show_json_validation = true;
2126                     *self.timetable_file_info.validation_status.lock()
2127                         .unwrap() =
2128                             TimetableValidationStatus::Ready;
2129                     self.selected_student = None;
2130                 }
2131             }
2132         }
2133         ui.add_enabled_ui(
2134             current_validation_status ==
2135             TimetableValidationStatus::Successful,
2136             |ui| {
2137                 if
2138                     ui
2139                         .button("Baseline benchmark")
2140                         .on_hover_text(
2141                             "Generate shortest routes for all
2142                             students as a baseline plan.");
2143                         .on_disabled_hover_text("Import a timetable
2144                             first.");
2145             );
2146         ui.add_enabled_ui(
2147             current_path_status == PathGenerationStatus::Successful,
2148             |ui| {
2149                 if
2150                     ui
2151                         .button("Calculate congestion")
2152                         .on_disabled_hover_text("Calculate routes
2153                             first.");
2154             );
2155         self.show_congestion_window = true;
2156         *self.congestion_status.lock().unwrap() =
2157             CongestionStatus::Ready;
2158     });
2159 }
```

```

2157         }
2158     }
2159     );
2160     ui.add_enabled_ui(
2161         current_congestion_status ==
2162         CongestionStatus::Successful,
2163         |ui| {
2164             if
2165                 ui
2166                     .button("Optimize routes")
2167                     .on_disabled_hover_text("Calculate congestion
2168 first.")
2169         iteration process."]
2170     {
2171         self.show_optimization_window = true;
2172         *self.optimization_status.lock().unwrap() =
2173             OptimizationStatus::ParamInput;
2174     }
2175 );
2176 ui.separator();
2177 ComboBox::from_label("Route display mode")
2178     .selected_text(format!("{}", self.path_display))
2179     .show_ui(ui, |ui| {
2180         ui.selectable_value(
2181             &mut self.path_display,
2182             PathDisplay::Shortest,
2183             "Shortest route"
2184         );
2185         ui.add_enabled_ui(
2186             self.student_routes_optimized.lock().unwrap()
2187             .is_some(),
2188             |ui| {
2189                 ui.selectable_value(
2190                     &mut self.path_display,
2191                     PathDisplay::Optimized,
2192                     "Optimized route"
2193                 );
2194             });
2195     });
2196 ComboBox::from_label("Student")
2197     .selected_text(self.selected_student.clone().unwrap_or("
2198 -".to_owned()))
2199     .show_ui(ui, |ui| {
2200         ui.add(
2201             egui::TextEdit
2202                 ::singleline(&mut self.student_number_search)
2203                 .hint_text("Search")
2204             );
2205         ui.separator();
2206         for student in self.student_list.lock().unwrap()
2207             .iter() {
2208             if student.contains(&self.student_number_search)
2209                 ui.selectable_value(
2210                     &mut self.selected_student,
2211                     Some(student.to_owned()),

```



```

2264             if let Some(file) = file {
2265                 let mut file = File::create(file).unwrap();
2266                 let student_routes =
2267                     self.student_routes_optimized.lock().unwrap();
2268                     .unwrap();
2269                     student_routes.unwrap();
2270                     .unwrap();
2271                     }
2272                     });
2273                     ui.separator();
2274                     ui.heading("Floor view");
2275                     ui.horizontal(|ui| {
2276                         if ui.toggle_value(&mut self.selected_floor[0], "All")
2277                             .clicked() {
2278                             self.selected_floor_index = 0;
2279                             for i in 1..=8 {
2280                                 self.selected_floor[i] = false;
2281                             }
2282                             if !self.selected_floor.contains(&true) {
2283                                 self.selected_floor[0] = true;
2284                             }
2285                             for i in 2..=8 {
2286                                 if
2287                                     ui
2288                                         .toggle_value(&mut self.selected_floor[i],
2289                                         .clicked())
2290                                         {
2291                                             self.selected_floor_index = i;
2292                                             for j in 0..=8 {
2293                                                 if i != j {
2294                                                     self.selected_floor[j] = false;
2295                                                 }
2296                                             }
2297                                             if !self.selected_floor.contains(&true) {
2298                                                 self.selected_floor[i] = true;
2299                                             }
2300                                         }
2301                                         }
2302                                         );
2303                                         ui.add(
2304                                             Slider::new(&mut self.inactive_brightness, 32..=255)
2305                                             .text(
2306                                                 "Inactive floor brightness"
2307                                             )
2308                                         );
2309                                         ui.separator();
2310                                         ui.horizontal(|ui| {
2311                                             if ui.selectable_label(!self.show_congestion, "Show
2312                                         paths").clicked() {
2313                                             self.show_congestion = false;
2314                                         }
2315                                         if ui.selectable_label(self.show_congestion, "Show
2316                                         congestion").clicked() {
2317                                             self.show_congestion = true;
2318                                         }

```

```

2316     });
2317     if self.show_congestion {
2318         ui.checkbox(&mut self.show_congestion_path, "Show path
2319         congestion");
2320         ui.checkbox(&mut self.show_congestion_point, "Show node
2321         congestion");
2322         ui.heading("Legend");
2323         egui::Grid
2324             ::new("congestion_legend")
2325             .num_columns(2)
2326             .show(ui, |ui| {
2327                 ui.label(RichText::new("●"))
2328                 .color(congestion_color_scale(0)));
2329                 ui.label("No students");
2330                 ui.end_row();
2331                 ui.label(RichText::new("●"))
2332                 .color(congestion_color_scale(1)));
2333                 ui.label("1-20 students");
2334                 ui.end_row();
2335                 ui.label(RichText::new("●"))
2336                 .color(congestion_color_scale(21)));
2337                 ui.label("21-50 students");
2338                 ui.end_row();
2339                 ui.label(RichText::new("●"))
2340                 .color(congestion_color_scale(51)));
2341                 ui.label("51-100 students");
2342                 ui.end_row();
2343                 ui.label(RichText::new("●"))
2344                 .color(congestion_color_scale(101)));
2345                 ui.label("101-200 students");
2346                 ui.end_row();
2347                 ui.label(RichText::new("●"))
2348                 .color(congestion_color_scale(201)));
2349                 ui.label("201-400 students");
2350                 ui.end_row();
2351                 ui.label(RichText::new("●"))
2352                 .color(congestion_color_scale(401)));
2353                 ui.label("≥401 students");
2354                 ui.end_row();
2355             });
2356             ui.add(
2357                 Slider::new(&mut self.congestion_filter, 0..=400)
2358             .text(
2359                 "Minimum congestion"
2360             )
2361         );
2362     } else {
2363         ui.horizontal(|ui| {
2364             ui.vertical(|ui| {
2365                 ui.label("Active path color");
2366                 if ui.button("Reset").clicked() {
2367                     self.active_path_color =
2368                         Color32::from_rgb(0xec, 0x6f, 0x27);
2369                 }
2370             });
2371             color_picker::color_picker_color32(
2372                 ui,
2373                 &mut self.active_path_color,
2374                 color_picker::Alpha::Opaque
2375             );
2376         });
2377     });

```

```

2366             ui.separator();
2367
2368             ui.horizontal(|ui| {
2369                 ui.vertical(|ui| {
2370                     ui.label("Inactive path color");
2371                     if ui.button("Reset").clicked() {
2372                         self.inactive_path_color =
2373                             Color32::from_gray(0x61);
2374                     }
2375                 });
2376                 color_picker::color_picker_color32(
2377                     ui,
2378                     &mut self.inactive_path_color,
2379                     color_picker::Alpha::Opaque
2380                 );
2381             });
2382         });
2383     });
2384 });
2385
2386 CentralPanel::default().show(ctx, |ui| {
2387     if self.show_json_validation {
2388         self.show_json_validation_window(ctx, current_validation_status);
2389     }
2390
2391     if self.show_path_gen_window {
2392         self.show_path_generation_window(ctx, current_path_status);
2393     }
2394
2395     if self.show_timetable_window {
2396         self.show_timetable_window(ctx);
2397     }
2398
2399     if self.show_congestion_window {
2400         self.show_congestion_window(ctx, current_congestion_status);
2401     }
2402     if self.show_pi_window {
2403         self.show_pi_window(ctx);
2404     }
2405     if self.show_optimization_window {
2406         self.show_optimization_window(ctx, current_optimization_status);
2407     }
2408
2409     // Paths
2410
2411     let path_list: Vec<String> = if self.selected_student.is_some() {
2412         let student_number = self.selected_student.clone().unwrap();
2413         let student_routes = if self.path_display ==
PathDisplay::Optimized {
2414             self.student_routes_optimized.lock().unwrap().clone()
2415         } else {
2416             self.student_routes_shortest.lock().unwrap().clone()
2417         };
2418         if let Some(student_routes) = student_routes {
2419             student_routes[&student_number][&self.selected_day][&
self.selected_period]
2420                 .split(' ')
2421                 .collect::<Vec<&str>>()

```

```

2422             .iter()
2423                 .map(|s| (*s).to_owned())
2424                 .collect::<Vec<String>>()
2425         } else {
2426             vec![]
2427         }
2428     } else {
2429         vec![]
2430     };
2431
2432     let mut segments: Vec<&[i32; 3]> = vec![];
2433
2434     for i in &path_list {
2435         if self.projection_coords.contains_key(i) {
2436             segments.push(&self.projection_coords[i]);
2437         }
2438     }
2439
2440     // Paths window
2441     if self.show_path_window {
2442         Window::new("Path")
2443             .open(&mut self.show_path_window)
2444             .show(ctx, |ui| {
2445                 let mut path_string = String::new();
2446                 for i in &path_list {
2447                     path_string.push_str(i);
2448                     path_string.push_str(" → ");
2449                 }
2450                 path_string.pop();
2451                 path_string.pop();
2452                 path_string.pop();
2453                 ui.label(path_string);
2454             });
2455     }
2456
2457     // Import textures if uninitialized
2458
2459     let mut textures: Vec<TextureHandle> = Vec::new();
2460     for i in 2..=8 {
2461         let texture_cur: &TextureHandle = self.textures[i]
2462             .get_or_insert_with(|| {
2463                 ui.ctx().load_texture(
2464                     format!("texture-floor-projection-{i}F"),
2465                     load_image_from_path(
2466                         Path::new(
2467                             format!(
2468                                 "assets/projection-transparent/projection_{i}
2469 F.png"
2470                         ).as_str()
2471                         )
2472                     )
2473                 );
2474                 textures.push(texture_cur.clone());
2475             })
2476             .unwrap(),
2477             Default::default()
2478         );
2479     }
2480     let desired_size = ui.available_size_before_wrap();
2481     if desired_size.y < (desired_size.x / 2243.0) * (1221.0 + 350.0) {

```

```

2478         ui.label("▲ There may not be enough space to display the floor
2479 plan.");
2480     }
2481     let (_id, rect) = ui.allocate_space(desired_size);
2482     let scale = rect.width() / 2243.0;
2483
2484     // Paint floor projections
2485
2486     let current_floor_z = if self.selected_floor_index == 0 {
2487         0
2488     } else {
2489         ((self.selected_floor_index - 2) * 50) as i32
2490     };
2491
2492     if !self.show_congestion {
2493         for (i, point) in segments.iter().enumerate() {
2494             if i != 0 {
2495                 ui.painter().circle_filled(convert_pos(&rect, point,
2496 scale), 4.0, if
2497                     self.selected_floor_index == 0 ||
2498                     (current_floor_z >= point[2].min(segments[i - 1][2])
2499                     &&
2500                     current_floor_z <= point[2].max(segments[i - 1]
2501 [2])))
2502             {
2503                 self.active_path_color
2504             } else {
2505                 self.inactive_path_color
2506             });
2507             ui.painter().line_segment(
2508                 [
2509                     convert_pos(&rect, segments[i - 1], scale),
2510                     convert_pos(&rect, point, scale),
2511                 ],
2512                 if
2513                     self.selected_floor_index == 0 ||
2514                     (current_floor_z >= point[2].min(segments[i - 1]
2515 [2]) &&
2516                     current_floor_z <= point[2].max(segments[i - 1][2]))
2517                 {
2518                     Stroke::new(4.0, self.active_path_color)
2519                 } else {
2520                     Stroke::new(4.0, self.inactive_path_color)
2521                 }
2522             );
2523         } else {
2524             ui.painter().circle_filled(convert_pos(&rect, point,
2525 scale), 4.0, if
2526                     self.selected_floor_index == 0 ||
2527                     current_floor_z == point[2].min(segments[i][2])
2528                 {
2529                     self.active_path_color
2530                 } else {
2531                     self.inactive_path_color
2532                 });
2533             }
2534         }
2535     } else {
2536         if self.show_congestion_path {

```

```

2531         if
2532             self.path_display == PathDisplay::Optimized &&
2533             self.student_routes_optimized.lock().unwrap().is_some()
2534         {
2535             for ((node1, node2), congestion) in
2536                 self.congestion_path_data_opt
2537                     .lock()
2538                     .unwrap()
2539                     .clone()
2540                     .get(&self.selected_day)
2541                     .unwrap()
2542                     .get(&self.selected_period)
2543                     .unwrap() {
2544                         if node1 == "G" || node2 == "G" || congestion < &
2545                             continue;
2546                         }
2547                         let node1_pos = self.projection_coords[node1];
2548                         let node2_pos = self.projection_coords[node2];
2549                         if
2550                             self.selected_floor_index == 0 ||
2551                             (current_floor_z >= node1_pos[2].min(node2_pos[2])
2552                             ) &&
2553                             .max(node2_pos[2]))
2554                         {
2555                             ui.painter().line_segment(
2556                                 [
2557                                     convert_pos(&rect, &node1_pos, scale),
2558                                     convert_pos(&rect, &node2_pos, scale),
2559                                     ],
2560                                     Stroke::new(4.0,
2561                                     congestion_color_scale(*congestion))
2562                                     );
2563                         }
2564                         } else {
2565                             for ((node1, node2), congestion) in
2566                                 self.congestion_path_data
2567                                     .lock()
2568                                     .unwrap()
2569                                     .clone()
2570                                     .get(&self.selected_day)
2571                                     .unwrap()
2572                                     .get(&self.selected_period)
2573                                     .unwrap() {
2574                                         if node1 == "G" || node2 == "G" || congestion < &
2575                                             continue;
2576                                         }
2577                                         let node1_pos = self.projection_coords[node1];
2578                                         let node2_pos = self.projection_coords[node2];
2579                                         if
2580                                             self.selected_floor_index == 0 ||
2581                                             (current_floor_z >= node1_pos[2].min(node2_pos[2]
2582                                             ) &&
2583                                             .max(node2_pos[2]))
2584                                         {
2585                                             ui.painter().line_segment(
2586                                                 [

```



```

2638             convert_pos(&rect, coords, scale),
2639             4.0,
2640             congestion_color_scale(*congestion)
2641         );
2642     }
2643 }
2644 }
2645 }
2646 }
2647
2648 for (i, texture) in textures.iter().enumerate().take(7) {
2649     let rect = Rect::from_min_size(
2650         rect.min,
2651         emath::vec2(rect.width(), rect.width() /
texture.aspect_ratio())
2652             .translate(emath::vec2(0.0, ((7 - i) as f32) * 50.0 * scale));
2653
2654     ui.painter().image(
2655         texture.into(),
2656         rect,
2657         Rect::from_min_max(pos2(0.0, 0.0), pos2(1.0, 1.0)),
2658         if self.selected_floor[0] || self.selected_floor[i + 2] {
2659             Color32::WHITE
2660         } else {
2661             Color32::from_gray(self.inactive_brightness)
2662         }
2663     );
2664 }
2665 // Special case: the floor is selected, so needs to be repainted last
2666 if self.selected_floor_index != 0 {
2667     let texture = textures[self.selected_floor_index - 2].clone();
2668     let rect = Rect::from_min_size(
2669         rect.min,
2670         emath::vec2(rect.width(), rect.width() /
texture.aspect_ratio())
2671             .translate(
2672                 emath::vec2(0.0, ((7 - (self.selected_floor_index - 2)) as
f32) * 50.0 * scale)
2673             );
2674
2675     ui.painter().image(
2676         (&texture).into(),
2677         rect,
2678         Rect::from_min_max(pos2(0.0, 0.0), pos2(1.0, 1.0)),
2679         Color32::WHITE
2680     );
2681 }
2682 });
2683 }
2684 }
2685
2686 fn load_image_from_path(path: &Path) → Result<ColorImage, image::ImageError> {
2687     let image = image::io::Reader::open(path)?.decode()?;
2688     let size = [image.width() as _, image.height() as _];
2689     let image_buffer = image.to_rgba8();
2690     let pixels = image_buffer.as_flat_samples();
2691     Ok(ColorImage::from_rgba_unmultiplied(size, pixels.as_slice()))
2692 }
2693

```

```

2694 // Converts 3D coordinates in projection-coords.yaml to 2D coordinates on
2695 // screen.
2696 fn convert_pos(rect: &Rect, pos: &[i32; 3], scale: f32) → emath::Pos2 {
2697     /// Projection angle of the floor plan (radians)
2698     const ANGLE: f32 = PI / 6.0;
2699
2700     (
2701         emath::vec2(
2702             rect.left() + 25.0 * ANGLE.cos() * scale,
2703             rect.top() + (50.0 + 350.0 + 25.0 * ANGLE.sin()) * scale
2704         ) +
2705         emath::vec2(
2706             ((pos[0] as f32) * ANGLE.cos() + (pos[1] as f32) * ANGLE.cos()) *
2707             scale,
2708             ((pos[0] as f32) * ANGLE.sin() - (pos[1] as f32) * ANGLE.sin() -
2709             (pos[2] as f32)) *
2710                 scale
2711         )
2712     ).to_pos2()
2713 }
2714
2715 fn convert_day_of_week(day: u32) → String {
2716     (
2717         match day {
2718             1 => "Monday",
2719             2 => "Tuesday",
2720             3 => "Wednesday",
2721             4 => "Thursday",
2722             5 => "Friday",
2723             _ => "Unknown",
2724         }
2725     ).into()
2726 }
2727
2728 fn convert_periods(index: usize) → String {
2729     (
2730         match index {
2731             0 => "Before P1",
2732             1 => "P1-P2",
2733             2 => "P2-P3",
2734             3 => "P3-P4",
2735             4 => "P4-P5",
2736             5 => "P5-P6",
2737             6 => "P6-Lunch",
2738             7 => "Lunch-P7",
2739             8 => "P7-P8",
2740             9 => "P8-P9",
2741             10 => "P9-P10",
2742             11 => "After P10",
2743             _ => "Unknown",
2744         }
2745     ).into()
2746 }
2747
2748 fn congestion_color_scale(congestion: u32) → Color32 {
2749     match congestion {
2750         0 => Color32::from_rgb(0x61, 0x61, 0x61),
2751         1..=20 => Color32::from_rgb(0x00, 0x7a, 0xf5),
2752         21..=50 => Color32::from_rgb(0x14, 0xae, 0x52),
2753         51..=100 => Color32::from_rgb(0xff, 0xc1, 0x07),

```

```
2750     101..=200 => Color32::from_rgb(0xec, 0x6f, 0x27),
2751     201..=400 => Color32::from_rgb(0xe4, 0x37, 0x48),
2752     _ => Color32::from_rgb(0x91, 0x54, 0xff),
2753 }
2754 }
2755
2756 fn congestion_range_index(congestion: u32) → usize {
2757     match congestion {
2758         0 => 0,
2759         1..=20 => 1,
2760         21..=50 => 2,
2761         51..=100 => 3,
2762         101..=200 => 4,
2763         201..=400 => 5,
2764         _ => 6,
2765     }
2766 }
2767 }
```

## optiway/src/lib.rs

```
1 | #![warn(clippy::all, rust_2018_idioms)]
2 |
3 | mod app;
4 | pub use app::OptiWayApp;
5 | mod app_init;
6 | pub use app_init::{ setup_custom_fonts, setup_custom_styles };
7 | pub mod md_icons;
8 | 
```

## optiway/src/main.rs

```
1  #![warn(clippy::all, rust_2018_idioms)]
2  #![cfg_attr(not(debug_assertions), windows_subsystem = "windows")]
3
4  #[cfg(compile_env = "bundle")]
5  use std::env;
6
7  #[cfg(not(target_arch = "wasm32"))]
8  fn main() → eframe::Result<()> {
9      #[cfg(compile_env = "bundle")]
10     {
11         env::set_current_dir(env::current_exe().unwrap().parent().unwrap())
12         .unwrap();
13     }
14     env_logger::init();
15
16     let native_options = eframe::NativeOptions::default();
17     eframe::run_native(
18         "OptiWay",
19         native_options,
20         Box::new(|cc| {
21             // egui_extras::install_image_loaders(&cc.egui_ctx);
22             Box::new(optiway::OptiWayApp::new(cc))
23         })
24     )
25 }
26
27 #[cfg(target_arch = "wasm32")]
28 fn main() {
29     eframe::WebLogger::init(log::LevelFilter::Debug).ok();
30
31     let web_options = eframe::WebOptions::default();
32
33     wasm_bindgen_futures::spawn_local(async {
34         eframe::WebRunner
35             ::new()
36             .start(
37                 "the_canvas_id",
38                 web_options,
39                 Box::new(|cc| Box::new(optiway::OptiWayApp::new(cc)))
40             ).await
41             .expect("failed to start eframe");
42     });
43 }
```

## optiway/build.rs

```
1 | fn main() {
2 |     let env = std::env::var("OPTIWAY_COMPILE_ENV").unwrap_or("default".to_string());
3 |     println!("cargo:rerun-if-env-changed=OPTIWAY_COMPILE_ENV");
4 |     println!("cargo:rustc-cfg=compile_env=\"{}\"", env);
5 |
6 | }
```

## optiway/Cargo.toml

```
1 [package]
2 name = "optiway"
3 version = "0.1.0"
4 edition = "2021"
5
6 # See more keys and their definitions at https://doc.rust-
7 lang.org/cargo/reference/manifest.html
8
9 [dependencies]
10 egui = "0.22.0"
11 eframe = { version = "0.22.0", default-features = false, features = [
12     # "accesskit",      # Make egui compatible with screen readers. NOTE: adds a lot
13     # of dependencies.
14     "default_fonts",   # Embed the default egui fonts.
15     "glow",            # Use the glow rendering backend. Alternative: "wgpu".
16     # "persistence",    # Enable restoring app state when restarting the app.
17   ] }
18 log = "0.4"
19 rfd = "0.12.0"
20 egui_extras = { version = "0.22.0" }
21 image = { version = "0.24", default-features = false, features = [
22     "jpeg",
23     "png",
24   ] }
25 serde = { version = "1.0", features = ["derive"] }
26 serde_yaml = "0.8"
27 serde_json = "1.0"
28 num-format = "0.4.4"
29
30 [target.'cfg(not(target_arch = "wasm32"))'.dependencies]
31 env_logger = "0.10"
32
33 [target.'cfg(target_arch = "wasm32")'.dependencies]
34 wasm-bindgen-futures = "0.4"
35
36 [profile.release]
37 opt-level = 2
38
39 [profile.dev.package."]
40 opt-level = 2
41
```

## optiway/Makefile.toml

```
1 [env]
2 G_PLUS_PLUS = "g++"
3
4 [tasks.floyd-build]
5 command = "${G_PLUS_PLUS}"
6 args = [
7     "../multi_agent_path_finding/floyd.cpp",
8     "-o",
9     "bin/floyd.out",
10    "-std=c++17",
11    "-O0",
12    "-Xlinker",
13    "-ld_classic",
14 ]
15
16 # Apple uses their new linker in macOS Sonoma 14... unfortunately the linker
17 # crashes in a lot of cases (see https://developer.apple.com/forums/thread/737707)
18 # So we use the old linker for now. ["-Xlinker", "-ld_classic"]
19
20 [tasks.floyd-release-build]
21 command = "${G_PLUS_PLUS}"
22 args = [
23     "../multi_agent_path_finding/floyd.cpp",
24     "-o",
25     "bin/floyd.out",
26     "-std=c++17",
27     "-O3",
28     "-Xlinker",
29     "-ld_classic",
30 ]
31
32 [tasks.floyd-release-build-x64]
33 command = "${G_PLUS_PLUS}"
34 args = [
35     "../multi_agent_path_finding/floyd.cpp",
36     "-o",
37     "bin/floyd.out",
38     "-std=c++17",
39     "-O3",
40     "-arch",
41     "x86_64",
42     "-Xlinker",
43     "-ld_classic",
44 ]
45
46 [tasks.optimization-build]
47 command = "${G_PLUS_PLUS}"
48 args = [
49     "../multi_agent_path_finding/multi-objective-agent.cpp",
50     "-o",
51     "bin/optimization.out",
52     "-std=c++17",
53     "-O0",
54     "-Xlinker",
55     "-ld_classic",
56 ]
```

```

56
57 [tasks.optimization-release-build]
58 command = "${G_PLUS_PLUS}"
59 args = [
60     "../multi_agent_path_finding/multi-objective-agent.cpp",
61     "-o",
62     "bin/optimization.out",
63     "-std=c++17",
64     "-O3",
65     "-Xlinker",
66     "-ld_classic",
67 ]
68
69 [tasks.optimization-release-build-x64]
70 command = "${G_PLUS_PLUS}"
71 args = [
72     "../multi_agent_path_finding/multi-objective-agent.cpp",
73     "-o",
74     "bin/optimization.out",
75     "-std=c++17",
76     "-O3",
77     "-arch",
78     "x86_64",
79     "-Xlinker",
80     "-ld_classic",
81 ]
82
83 [tasks.entry-run]
84 command = "cargo"
85 args = ["run"]
86
87 [tasks.entry-build]
88 command = "cargo"
89 args = ["build"]
90
91 [tasks.entry-release-build]
92 command = "cargo"
93 args = ["build", "--release"]
94
95 [tasks.entry-release-run]
96 command = "cargo"
97 args = ["run", "--release"]
98
99 [tasks.entry-bundle-build]
100 env = { OPTIWAY_COMPILE_ENV = "bundle" }
101 command = "cargo"
102 args = ["build", "--release", "-Z", "unstable-options", "--out-dir", "bundle/"]
103
104 [tasks.entry-bundle-build-x64]
105 env = { OPTIWAY_COMPILE_ENV = "bundle" }
106 command = "cargo"
107 args = [
108     "build",
109     "--release",
110     "-Z",
111     "unstable-options",
112     "--out-dir",
113     "bundle/",
```

```

114     "--target",
115     "x86_64-apple-darwin",
116 ]
117
118 [tasks.bundle-clean-dirs]
119 script_runner = "python"
120 script_extension = "py"
121 script = """
122 import shutil
123 import os
124 for folder in ['bundle', 'bundle/bin', 'bundle/assets']:
125     if not os.path.exists(folder):
126         os.makedirs(folder)
127     else:
128         shutil.rmtree(folder)
129         os.makedirs(folder)
130 """
131
132 [tasks.bundle-script]
133 script_runner = "python"
134 script_extension = "py"
135 script = """
136 import shutil
137 import os
138 shutil.copyfile('bin/floyd.out', 'bundle/bin/floyd.out')
139 shutil.copyfile('bin/optimization.out', 'bundle/bin/optimization.out')
140
141 for root, dirs, files in os.walk('assets'):
142     for file in files:
143         if file != '.DS_Store':
144             if not os.path.exists(os.path.join('bundle', root)):
145                 os.makedirs(os.path.join('bundle', root))
146             shutil.copyfile(os.path.join(root, file), os.path.join('bundle',
147 os.path.join(root, file)))
148 os.remove('bundle/liboptiway.rlib')
149 """
150
151 [tasks.bundle-chmod]
152 command = "chmod"
153 args = ["-R", "+x", "bundle/"]
154
155 [tasks.bundle]
156 dependencies = [
157     "bundle-clean-dirs",
158     "floyd-release-build",
159     "optimization-release-build",
160     "entry-bundle-build",
161     "bundle-script",
162     "bundle-chmod",
163 ]
164
165 [tasks.bundle-x64]
166 dependencies = [
167     "bundle-clean-dirs",
168     "floyd-release-build-x64",
169     "optimization-release-build-x64",
170     "entry-bundle-build-x64",
171     "bundle-script",

```

```
172     "bundle-chmod",
173 ]
174
175 [tasks.run]
176 dependencies = ["floyd-build", "optimization-build", "entry-run"]
177
178 [tasks.build]
179 dependencies = ["floyd-build", "optimization-build", "entry-build"]
180
181 [tasks.release]
182 dependencies = [
183     "floyd-release-build",
184     "optimization-release-build",
185     "entry-release-run",
186 ]
187
```

## optiway/rustfmt.toml

```
1 | max_width = 100
```

## scripts/coord-flatten.py

```
1 # coord-flatten.py
2 # Created by Michael Zhang
3
4 # This script flattens projection-coords.yaml file.
5
6
7 import yaml
8
9 with open("assets/projection-coords.yaml", "r") as f:
10     try:
11         source = yaml.safe_load(f)
12     except yaml.YAMLError as e:
13         print(e)
14
15 result = []
16
17 for floor_key in source:
18     for side_key in source[floor_key]:
19         if source[floor_key][side_key] != None:
20             for i in source[floor_key][side_key]:
21                 result[i] = source[floor_key][side_key][i]
22
23 with open("assets/projection-coords-flatten.yaml", "w") as f:
24     yaml.safe_dump(result, f)
25
```

## scripts/coord-generator.py

```
1 # coord-generator.py
2 # Created by Michael Zhang
3
4 # This script generates the x-coordinates of the nodes
5 # by calculating the midpoint using the rooms' lengths.
6
7 A_BLOCK = False
8 FIRST_OFFSET = 540
9 LAST_OFFSET = 40
10
11 rooms = []
12 lengths = []
13 with open('assets/projection-length.yaml', 'r') as f:
14     for line in f.readlines():
15         room, length = line.split(': ')
16         length = int(length)
17         rooms.append(room)
18         lengths.append(length)
19
20
21 if A_BLOCK:
22     print("A block. Total length:", sum(lengths) + FIRST_OFFSET)
23     if FIRST_OFFSET != 0:
24         print(
25             f"\033[1;34mNOTE: The first node offset is set to {FIRST_OFFSET}\033[0m")
26     if sum(lengths) + FIRST_OFFSET != 2290:
27         print("\033[1;33mWARNING: Total length is not 2290!\033[0m")
28     for i in rooms:
29         if i[0] == 'B':
30             print(
31                 "\033[1;33mWARNING: Using A block mode, but data contains B block"
32 rooms!\033[0m")
33             break
34     for i in range(len(rooms)):
35         print(
36             f'{rooms[i]}: {FIRST_OFFSET+int(sum(lengths[:i]) + lengths[i] / 2)}')
37 else:
38     print("B block. Total length:", sum(lengths))
39     if LAST_OFFSET != 0:
40         print(
41             f"\033[1;34mNOTE: The last node offset is set to {LAST_OFFSET}.\033[0m")
42     for i in rooms:
43         if i[0] == 'A':
44             print(
45                 "\033[1;33mWARNING: Using B block mode, but data contains A block"
46 rooms!\033[0m")
47             break
48     for i in range(len(rooms)):
49         print(
50             f'{rooms[i]}: {2290-LAST_OFFSET-int(sum(lengths[:i]) + lengths[i] / 2)}')
```

## scripts/crossing-paths.py

```
1 # coord-paths.py
2 # Created by Michael Zhang
3
4 import yaml
5
6 with open("assets/projection-coords-flatten.yaml", "r") as f:
7     try:
8         source = yaml.safe_load(f)
9     except yaml.YAMLError as e:
10         print(e)
11
12 room_1 = input("Enter room 1    >>> ")
13 crossing = input("Enter crossing >>> ")
14 room_2 = input("Enter room 2    >>> ")
15
16 if room_1 not in source:
17     print(f"Room {room_1} not found!")
18     exit(1)
19 if room_2 not in source:
20     print(f"Room {room_2} not found!")
21     exit(1)
22 if crossing not in source:
23     print(f"Crossing {crossing} not found!")
24     exit(1)
25
26 x1 = source[room_1][0]
27 x2 = source[room_2][0]
28 xc = source[crossing][0]
29
30 print(" - dist:", abs(x1 - xc))
31 print("   nodes:")
32 print("     -", room_1)
33 print("     -", crossing)
34 print("     type: 0")
35 print(" - dist:", abs(x2 - xc))
36 print("   nodes:")
37 print("     -", crossing)
38 print("     -", room_2)
39 print("     type: 0")
40
```

## scripts/path-converter.py

```
1 # path-converter.py
2 # Created by Michael Zhang
3
4 import yaml
5
6 with open("assets/paths.yaml", "r") as f:
7     try:
8         source = yaml.safe_load(f)
9     except yaml.YAMLError as e:
10         print(e)
11
12 for floor_key in source:
13     for side_key in source[floor_key]:
14         if source[floor_key][side_key] != None:
15             for i in source[floor_key][side_key]:
16                 print(i["nodes"][0], i["nodes"][1], i["dist"], i["type"])
17
18 print("""S2-1 G 999999 2
19 S2-3 G 999999 2
20 S2-4 G 999999 2
21 S2-5 G 999999 2
22 S2-6 G 999999 2
23 S2-7 G 999999 2""")
```

## scripts/path-generator.py

```
1 # path-generator.py
2 # Created by Michael Zhang
3
4 # This script generates paths on the same floor on one block
5 # using data from assets/projection-coords.yaml
6
7 # Path types:
8 # 0: Normal path between two rooms / between a room and one end of a bridge
9 # 1: Bridge between two buildings
10 # 2: Normal staircases (e.g. S2-3)
11 # 3: Spiral staircases between two floors only (e.g. S2-2)
12 # 4: Other types of staircases (e.g. S4-6)
13
14 import yaml # well yeah I just love YAML ;)
15
16 result = []
17
18 with open("assets/projection-coords.yaml", "r") as f:
19     try:
20         source = yaml.safe_load(f)
21     except yaml.YAMLError as e:
22         print(e)
23
24 for floor_key in source:
25     result[floor_key] = []
26     for block in ["A", "B"]:
27         result[floor_key][block] = []
28         for room_idx in range(len(source[floor_key][block].keys()) - 1):
29             room_keys = list(source[floor_key][block].keys())
30             x1 = source[floor_key][block][room_keys[room_idx]][0]
31             x2 = source[floor_key][block][room_keys[room_idx + 1]][0]
32             dist = abs(x2 - x1)
33             result[floor_key][block].append(
34                 {"nodes": (room_keys[room_idx], room_keys[room_idx+1]), "dist": dist, "type": 0})
35
36 try:
37     with open("assets/paths.yaml", "r") as f:
38         response = input("assets/paths.yaml already exists. Overwrite? (y/n) ")
39         if response == "y":
40             print("Overwriting assets/paths.yaml...")
41         else:
42             print("Aborted.")
43             exit()
44 except FileNotFoundError:
45     pass
46
47 with open("assets/paths.yaml", "w") as f:
48     yaml.safe_dump(result, f)
49
```

## timetable\_generation/timetable\_generator.py

```
1 """
2     Return File Structure (JSON):
3 {
4     student_number1 (string): {
5         day (string): {
6             period (string): room (string)
7         }
8         ...
9     }
10 }
11
12 Example:
13 {
14     "21447": {
15         "1": {
16             "1": "531",
17             "2": "531",
18             "3": "430",
19             "4": "430"
20             ...
21         },
22         "2": {...},
23         ...
24     },
25     "21448": {
26         "1": {
27             "1": "210",
28             "2": "210",
29             "3": "322",
30             "4": "322"
31             ...
32         },
33         "2": {...},
34         ...
35     }
36 }
37 """
38
39 import random
40 import copy
41 import json
42
43
44 class TimetableGenerator:
45     def __init__(self):
46         # Init Data #
47         # the frequency for elective classes
48         self.glevel_elective_freq = {
49             "Extra": 12,
50             "Bus": 18,
51             "Hum": 14,
52             "Enr": 6,
53             "Lan": 5,
54             "Sci": 25
55         }
56         # the dict to map class to floor number
```

```

57     self.floor_map = {
58         "Extra": 1,
59         "Bus": 2,
60         "Hum": 3,
61         "Enr": 4,
62         "Lan": 5,
63         "Chi": 5,
64         "Eng": 5,
65         # does not account for sci as it covers three floors
66     }
67     # all the available classrooms
68     self.avail_rooms = {
69         2: set(range(201, 242)) - {203, 209, 213, 218, 229, 230, 232, 234,
70         236, 237, 238, 240},
71         3: set(range(301, 334)) - {303, 304, 309, 317, 321, 322, 328},
72         4: set(range(401, 434)) - {403, 404, 412, 413, 418, 420, 422, 428},
73         5: set(range(501, 535)) - {503, 512, 517, 518, 520, 527, 528, 529},
74         6: set(range(601, 632)) - {603, 614, 615, 620, 623, 625, 626, 627},
75         7: set(range(701, 734)) - {703, 714, 716, 720, 723, 725, 727, 729,
76         730, 732},
77         8: set(range(801, 822)) - {808, 810, 813, 816, 818, 820}
78     }
79     # the dicts to map glevel schedule set to period number
80     # formate: "<day><1st period><2nd period>"
81     glevel_core_set_map = {
82         0: ["1_7_8", "4_9_10"],
83         1: ["1_9_10", "3_3_4"],
84         2: ["2_1_2", "5_3_4"],
85         3: ["2_3_4", "5_7_8"]
86     }
87     glevel_elect_set_map = {
88         0: ["1_1_2", "4_3_4"],
89         1: ["1_3_4", "2_9_10"],
90         2: ["2_7_8", "4_7_8"],
91         3: ["3_1_2", "5_1_2"]
92     }
93     self.glevel_set_map = [glevel_core_set_map, glevel_elect_set_map]
94     # the currently filled rooms
95     self.filled_rooms = [
96         dict(zip(range(2, 9), [set() for _ in range(2, 9)])) for _ in range(9)
97     ]
98     self.time_table = {}
99
100    # map rooms into different buildings
101    @staticmethod
102    def map_building(room):
103        if 200 < room < 221 or 300 < room < 320 or 400 < room < 421 or 500 < room
104        < 521 or 600 < room < 618 or 700 < room < 717 or 800 < room < 811:
105            return "A"
106        else:
107            return "B"
108
109    def generate_g_level(self, grade):
110        """
111            Generate G-level classes
112            1: Find the total number of classes for each of the categories: Extra(1F),
113            Bus(2F), Hum(3F), Enr(4F), Lan(5F), Sci(6F)
114                1.1 assume no 1-period classes → 380 2-period classes per week → 360
115            non-PE 2-period classes (500 / 25 = 20)
116                1.2 180 non-repetitive non-PE courses (1 course = 2 classes per week)

```

```

111     1.3 100 non-repetitive elective courses (core courses: chinese, eng,
112     math, chem) → true as everyone has 5 electives
113         1.3.1 80 clean extra courses (there are one more 2-period english
114         and enrichment class)
115             2: distribute 80 courses into Extra, Bus, Hum, Enr, Lan, Sci following
116             common-sense weightings (dict "elective_freq")
117                 2.1 Extra(art/drama/pe/music): 12
118                 2.2 Bus (econ/bus): 18
119                 2.3 Hum (geo+his+gp): 14
120                 2.4 Enr (cs): 6
121                 2.5 Lan (fren/span/ja): 5
122                 2.6 Sci (bio+phy): 25
123             3: distribute timeslots for each course
124                 3.1 assign fixed scheduling
125                     - SCIE has fixed scheduling - it separates classes into different
126                     timeslot couples
127                         e.g., courses at Monday P1 would repeat at Wednesday P1
128                         - 19 2-periods in a week, 18 non-PE, 2-periods in a week, 9
129                     schedule sets,
130                         - 8 clean schedule sets (without enrichment and another eng)
131                         - schedule sets for core and electives are separated
132                         - hence, distribute each class into these sets
133             3.2 define schedule sets
134                 - 4 core schedule sets (Enr, Chi, Eng, Sci)
135                     1. Mon78, Thu90
136                     2. Mon90, Wed34
137                     3. Tue12, Fri34
138                     4. Tue34, Fri78
139                     - 4 elective schedule sets (Extra, Bus, Hum, Lan)
140                         1: Mon12, Thu34
141                         2. Mon34, Tue90
142                         3. Tue78, Thu78
143                         4. Wed12, Fri12
144             4: distribute the rooms for each course
145                 4.1 the rooms available for each floor is stored in the dict "
146                 avail_rooms"
147                     4.2 limitations: courses at each scheduling set should not have the
148                     same room
149             5: distribute each student's list of courses taken
150             6: project back to generate the timetable
151
152     :param: grade: either G1 or G2
153     :return: timetable as a JSON
154     """
155
156     # Procedures #
157     # 3 distribute timeslots for each course, each slot only needs 20 courses
158     core_slots = [[] for _ in range(4)]
159     # assign the core classes
160     for core in range(20):
161         core_classes = ["Enr", "Chi", "Eng", "Sci"]
162         for i in range(4):
163             core_class = random.choice(core_classes)
164             core_slots[i].append(core_class + "_core_" + str(core))
165             core_classes.remove(core_class)
166     # assign the elective classes
167     elect_slots = [[] for _ in range(4)]
168     avail_slots = list(range(4))
169     for k, v in self.glevel_elective_freq.items():
170         for i in range(v):
171             slot = random.choice(avail_slots)

```

```

164         elect_slots[slot].append(k + "_elect_" + str(i))
165         if len(elect_slots[slot]) == 20:
166             avail_slots.remove(slot)
167
168     # 4 distribute the rooms for each course
169     if grade == "G1":
170         slots = core_slots + elect_slots
171     else:
172         slots = elect_slots + core_slots
173     rooms = {}
174     filled_rooms = [] # track the filled rooms for each floor for each slot
175     for index, slot in enumerate(slots):
176         filled_room = dict(zip(range(2, 9), [set() for _ in range(2, 9)]))
177         for course in slot:
178             if course[:3] == "Sci": # sci courses are distributed to three
floors
179                 # TODO: change to greedy floor
180                 floor = random.choice([6, 7, 8])
181             else:
182                 floor = self.floor_map[course.split("_")[0]]
183                 if floor == 1: # if it is extra, just assign G
184                     rooms[course] = "G"
185                     continue
186                 # divide by the current filled rooms, use mod because the core
slots for g1 are elective slots for g2
187                 room = random.choice(list(
188                     self.avail_rooms[floor] - filled_room[floor] -
self.filled_rooms[(index + 4) % 8][floor]))
189                 rooms[course] = self.map_building(room) + str(room)
190                 filled_room[floor].add(room)
191                 filled_rooms.append(filled_room)
192
193     # update the global filled_rooms
194     for i in range(8):
195         for j in range(2, 9):
196             self.filled_rooms[i][j].update(filled_rooms[i][j])
197             # [514, 519, 521, 523, 530, 531, 532, 533, 534, 502, 504, 508,
511]
198
199     # 5 distribute each student's list of courses taken
200     if grade == "G1":
201         offset = 23000
202     else:
203         offset = 22000
204     students = dict(zip(range(offset+1, offset+501),
205                         [[] for _ in range(500)]))
206     # assign each student to a core class (#0 - 19)
207     # track the number of students in each core class
208     core_counts = [0 for _ in range(20)]
209     cores = list(range(20))
210     for student in students:
211         core = random.choice(cores)
212         students[student].extend(
213             [f"Enr_core_{core}", f"Chi_core_{core}", f"Eng_core_{core}", f"Sci_core_{core}"])
214         core_counts[core] += 1
215         if core_counts[core] == 25:
216             cores.remove(core)
217
218     elect_slots_copy = copy.deepcopy(elect_slots)

```

```

219     # assign each student to an elective class
220     # track the filled courses for each slot
221     filled_courses = {
222         i: dict(zip(elect_slots_copy[i], [0] * 20)) for i in range(4)}
223     for student in students:
224         for i, slot in enumerate(elect_slots_copy):
225             course = random.choice(slot)
226             students[student].append(course)
227             filled_courses[i][course] += 1
228             if filled_courses[i][course] == 25:
229                 elect_slots_copy[i].remove(course)
230
231     # 6 projection back
232     timetable = {str(i): {str(j): {str(k): "G" for k in range(1, 11)}
233                             for j in range(1, 6)} for i in range(offset+1,
234 offset+501)}
235     # additional slot to hold the extra english and PE class
236     filled_rooms.append(
237         dict(zip(range(2, 9), [set() for _ in range(2, 9)])))
238     if grade == "G1":
239         core_set = 0
240         elect_set = 1
241     else:
242         core_set = 1
243         elect_set = 0
244     for k, student in students.items():
245         for course in student[:4]: # for the core classes
246             for index, slot in enumerate(core_slots):
247                 if course in slot:
248                     day, p1, p2 = self.glevel_set_map[core_set][index][0]
249                                     "_")
250                     timetable[str(k)][day][p1] = rooms[course]
251                     timetable[str(k)][day][p2] = rooms[course]
252                     day, p1, p2 = self.glevel_set_map[core_set][index][1]
253                                     "_")
254                     timetable[str(k)][day][p1] = rooms[course]
255                     timetable[str(k)][day][p2] = rooms[course]
256     # handle the extra english and PE class to spare slots
257     if course.split("_")[0] == 'Eng' and int(course.split("_")[-1]) <=
258         9:
259             timetable[str(k)]['4']['1'] = rooms[course]
260             timetable[str(k)]['4']['2'] = rooms[course]
261             timetable[str(k)]['3']['7'] = 'G'
262             timetable[str(k)]['3']['8'] = 'G'
263             filled_rooms[-1][5].add(rooms[course])
264     elif course.split("_")[0] == 'Eng' and int(course.split("_")[-1]) >
265         9:
266             timetable[str(k)]['4']['1'] = 'G'
267             timetable[str(k)]['4']['2'] = 'G'
268             timetable[str(k)]['3']['7'] = rooms[course]
269             # TODO: place G1 and G2 with un-collided English rooms
270             timetable[str(k)]['3']['8'] = rooms[course]
271             # add g2 extra classes
272             filled_rooms[-1][5].add(rooms[course])
273     for course in student[4:]: # for the elective classes
274         for index, slot in enumerate(elect_slots):
275             if course in slot:

```

```

273     .split(
274         " _")
275     timetable[str(k)][day][p1] = rooms[course]
276     timetable[str(k)][day][p2] = rooms[course]
277     day, p1, p2 = self.glevel_set_map[elect_set][index][1]
278     .split(
279         " _")
280     timetable[str(k)][day][p1] = rooms[course]
281     timetable[str(k)][day][p2] = rooms[course]
282
283     self.filled_rooms[8][5].update(filled_rooms[8][5])
284     self.time_table.update(timetable)
285
286     return timetable
287
288     def generate_as_al_level(self):
289         """
290             Generate AS-Level Classes
291             1: Find the total number of classes for each of the categories: Extra(1F),
292             Bus(2F), Hum(3F), Enr(4F), Lan(5F), Sci(6F)
293                 1.1 assume that each student take 1 Eng and 4 Electives → 5*3=15 2-
294                 periods per week for a student
295                     1.2 5*500/25=100 2-periods courses in total
296             2: distribute 100 courses into Extra, Bus, Hum, Enr, Lan, Sci following
297             common-sense weightings (dict "as_freq")
298                 2.1 Extra (art/drama/pe/music): 4
299                 2.2 Bus (econ/acc): 10
300                 2.3 Hum (geo/his/psy): 10
301                 2.4 Mat (mat/further/cs): 23
302                 2.5 Eng (lit/lan/gc/3rd lan): 23
303                 2.6 Sci (bio/phy/chem): 30
304             3: distribute timeslots for each course
305                 3.1 assign fixed scheduling
306                     - each slot accounts for 3 2-periods in AS
307                     - there are in total 6 slots for AS → don't need to fill in every
308             slot
309                 3.2 define schedule sets
310                     1. Mon12, Tue78, Thu34
311                     2. Mon34, Tue90, Thu78
312                     3. Mon78, Wed12, Thu90
313                     4. Mon90, Wed34, Fri12
314                     5. Tue12, Wed78, Fri34
315                     6. Tue34, Thu12, Fri56
316             4: distribute the rooms for each course
317                 4.1 the rooms available for each floor is stored in the dict "
318             avail_rooms"
319                 4.2 limitations: courses at each scheduling set should not have the
320                     same room
321                         - should check "filled_rooms" when handling this
322                         - hence, process "filled_rooms" into a timetable format
323             5: distribute each student's list of courses taken
324             6: project back to generate the timetable
325
326
327     :return: timetable as a JSON
328     """
329
330     # dictionary for as course frequencies
331     as_freq = {
332         "Extra": 4,

```

```

325     "Bus": 10,
326     "Hum": 10,
327     "Enr": 23,
328     "Eng": 23,
329     "Sci": 30
330 }
331 # dictionary for as course scheduling sets
332 as_set_map = {
333     0: ["1_1_2", "2_7_8", "4_3_4"],
334     1: ["1_3_4", "2_9_10", "4_7_8"],
335     2: ["1_7_8", "3_1_2", "4_9_10"],
336     3: ["1_9_10", "3_3_4", "5_1_2"],
337     4: ["2_1_2", "3_7_8", "5_3_4"],
338     5: ["2_3_4", "4_1_2", "5_5_6"]
339 }
340 # Procedures #
341 # 3 distribute timeslots for each course, each slot only needs 20 courses
342 slots = [] for _ in range(6)
343 avail_slots = list(range(6))
344 for k, v in as_freq.items():
345     for i in range(v):
346         slot = random.choice(avail_slots)
347         slots[slot].append(k + "_" + str(i))
348         if len(slots[slot]) == 20:
349             avail_slots.remove(slot)
350
351 # 4 distribute the rooms for each course
352 # transform the g_level filled_rooms into a timetable format
353 filled_timetable = {str(j): {str(k): set()
354                                 for k in range(1, 11)} for j in range(1, 6)}
355 glevel_set_map = list(self.glevel_set_map[0].values(
356 )) + list(self.glevel_set_map[1].values())
357 # don't include the extra two slots at last
358 for index, slot in enumerate(self.filled_rooms[:8]):
359     for v in slot.values():
360         for room in v:
361             day, p1, p2 = glevel_set_map[index][0].split("_")
362             filled_timetable[day][p1].add(room)
363             filled_timetable[day][p2].add(room)
364             day, p1, p2 = glevel_set_map[index][1].split("_")
365             filled_timetable[day][p1].add(room)
366             filled_timetable[day][p2].add(room)
367 # process the last two extra slots
368 for index, slot in enumerate(self.filled_rooms[8:]):
369     for v in slot.values():
370         for room in v:
371             filled_timetable['3']['7'].add(int(room[1:]))
372             filled_timetable['3']['8'].add(int(room[1:]))
373             filled_timetable['4']['1'].add(int(room[1:]))
374             filled_timetable['4']['2'].add(int(room[1:]))
375
376 # 4 distribute the rooms for each course
377 rooms = {}
378 filled_rooms = [] # track the filled rooms for each floor for each slot
379 for index, slot in enumerate(slots):
380     day1, p11, p12 = as_set_map[index][0].split("_")
381     day2, p21, p22 = as_set_map[index][1].split("_")
382     day3, p31, p32 = as_set_map[index][2].split("_")

```

```

383     filled_room = dict(zip(range(2, 9), [set() for _ in range(2, 9)]))
384     for course in slot:
385         if course[:3] == "Sci": # sci courses are distributed to three
386             # TODO: change to greedy floor
387             floor = random.choice([6, 7, 8])
388         else:
389             floor = self.floor_map[course.split("_")][0]
390             if floor == 1: # if it is extra, just assign G
391                 rooms[course] = "G"
392                 continue
393             # divide by the current filled rooms, use mod because the core
394             # slots for g1 are elective slots for g2
395             room = random.choice(
396                 list(self.avail_rooms[floor] - filled_timetable[day1][p11] -
397                     filled_timetable[day2][p21] - filled_timetable[day3][p31]))
398             rooms[course] = self.map_building(room) + str(room)
399             filled_room[floor].add(room)
400             filled_rooms.append(filled_room)

400     # 5 distribute each student's list of courses taken
401     # generate the students' slots
402     slots_count = [[i, len(slot)*25] for i, slot in enumerate(slots)]
403     offset = 21000
404     students_slots = {i: [] for i in range(offset+1, offset+501)}
405     for student in students_slots:
406         slots_count = sorted(slots_count, reverse=True, key=lambda x: x[1])
407         slots_indices = [item[0] for item in slots_count[:5]]
408         students_slots[student] = slots_indices
409         for i in range(6):
410             if slots_count[i][0] in slots_indices:
411                 slots_count[i][1] -= 1

412     # print(students_slots)

413     # generate the students
414     students = {i: [] for i in range(offset+1, offset+501)}
415     slots_copy = copy.deepcopy(slots)
416     # track the filled courses for each slot
417     filled_courses = {
418         i: dict(zip(slots_copy[i], [0] * len(slots_copy[i]))) for i in
419             range(6)}
420     for student in students:
421         # print(student)
422         for i in students_slots[student]:
423             slot = slots_copy[i]
424             course = random.choice(slot)
425             students[student].append(course)
426             filled_courses[i][course] += 1
427             if filled_courses[i][course] == 25:
428                 slots_copy[i].remove(course)

429     # 5 generate the timetable
430     timetable = {str(i): {str(j): {str(k): "G" for k in range(1, 11)}}
431                     for j in range(1, 6)} for i in range(offset+1,
432 offset+501)}
433     for k, student in students.items():
434         for course in student:
435             for index, slot in enumerate(slots):
436                 if course in slot:

```

```

438             day, p1, p2 = as_set_map[index][0].split("_")
439             timetable[str(k)][day][p1] = rooms[course]
440             timetable[str(k)][day][p2] = rooms[course]
441             day, p1, p2 = as_set_map[index][1].split("_")
442             timetable[str(k)][day][p1] = rooms[course]
443             timetable[str(k)][day][p2] = rooms[course]
444             day, p1, p2 = as_set_map[index][2].split("_")
445             timetable[str(k)][day][p1] = rooms[course]
446             timetable[str(k)][day][p2] = rooms[course]
447
448         """Generate AL Level, written in the same function to avoid passing
449         filled_rooms which is hard for different set maps"""
450         """
451             Generate AL-Level Classes
452             1: Find the total number of classes for each of the categories: Extra(1F),
453             Bus(2F), Hum(3F), Enr(4F), Lan(5F), Sci(6F)
454                 1.1 assume that each student take 1 Eng and 3 Electives → 4*3=12 2-
455                 periods per week for a student
456                     1.2 4*500/25=80 2-periods courses in total
457             2: distribute 80 courses into Extra, Bus, Hum, Enr, Lan, Sci following
458             common-sense weightings (dict "al_freq")
459                 2.1 Extra (art/drama/pe/music): 4
460                 2.2 Bus (econ/acc): 12
461                 2.3 Hum (geo/his/psy): 10
462                 2.4 Mat (mat/further/cs): 15
463                 2.5 Eng (lit/lan/gc/3rd lan): 23
464                 2.6 Sci (bio/phy/chem): 20
465             3: distribute timeslots for each course
466                 3.1 assign fixed scheduling
467                     - each slot accounts for 3 2-periods in AL
468                     - the set map for AL is the same as that for AS
469                 3.2 define schedule sets
470                     1. Mon12, Tue78, Thu34
471                     2. Mon34, Tue90, Thu78
472                     3. Mon78, Wed12, Thu90
473                     4. Mon90, Wed34, Fri12
474                     5. Tue12, Wed78, Fri34
475                     6. Tue34, Thu12, Fri56
476             4: distribute the rooms for each course
477                 4.1 the rooms available for each floor is stored in the dict "
478                 avail_rooms"
479                 4.2 limitations: courses at each scheduling set should not have the
480                 same room
481                     - should check "filled_rooms" when handling this
482                     - hence, process "filled_rooms" into a timetable format
483             5: distribute each student's list of courses taken
484             6: project back to generate the timetable
485
486
487         :return: timetable as a JSON
488         """
489         al_freq = {
490             "Extra": 4,
491             "Bus": 12,
492             "Hum": 10,
493             "Enr": 15,
494             "Eng": 23,
495             "Sci": 20
496         }

```

```

492     # PROCEDURES #
493     # 3 distribute timeslots for each course, each slot only needs 20 courses
494     slots = [[] for _ in range(6)]
495     avail_slots = list(range(6))
496     for k, v in al_freq.items():
497         for i in range(v):
498             slot = random.choice(avail_slots)
499             slots[slot].append(k + " " + str(i))
500             if len(slots[slot]) == 20:
501                 avail_slots.remove(slot)
502
503     # 4 distribute the rooms for each course
504     rooms = {}
505     for index, slot in enumerate(slots):
506         day1, p11, p12 = as_set_map[index][0].split("_")
507         day2, p21, p22 = as_set_map[index][1].split("_")
508         day3, p31, p32 = as_set_map[index][2].split("_")
509         filled_room = dict(zip(range(2, 9), [set() for _ in range(2, 9)]))
510         for course in slot:
511             if course[:3] == "Sci": # sci courses are distributed to three
floors
512                 # TODO: change to greedy floor
513                 floor = random.choice([6, 7, 8])
514             else:
515                 floor = self.floor_map[course.split("_")[0]]
516                 if floor == 1: # if it is extra, just assign G
517                     rooms[course] = "G"
518                     continue
519                 # divide by the current filled rooms, use mod because the core
slots for g1 are elective slots for g2
520                 room = random.choice(
521                     list(self.avail_rooms[floor] - filled_timetable[day1][p11] -
filled_timetable[day2][p21]
522                     - filled_timetable[day3][p31] - filled_rooms[index]
523 [floor])) # minus the as time slots as well
524                 rooms[course] = self.map_building(room) + str(room)
525                 filled_room[floor].add(room)
526
527     # 5 distribute each student's list of courses taken
528     # generate the students' slots
529     slots_count = [[i, len(slot)*25] for i, slot in enumerate(slots)]
530     offset = 20000
531     students_slots = {i: [] for i in range(offset+1, offset+501)}
532     for student in students_slots:
533         slots_count = sorted(slots_count, reverse=True, key=lambda x: x[1])
534         slots_indices = [item[0] for item in slots_count[:4]]
535         students_slots[student] = slots_indices
536         for i in range(6):
537             if slots_count[i][0] in slots_indices:
538                 slots_count[i][1] -= 1
539
540     # print(students_slots)
541
542     # generate the students
543     students = {i: [] for i in range(offset+1, offset+501)}
544     slots_copy = copy.deepcopy(slots)
545     # track the filled courses for each slot
546     filled_courses = {
547         i: dict(zip(slots_copy[i], [0] * len(slots_copy[i]))) for i in
range(6)}

```

```

547     for student in students:
548         # print(student)
549         for i in students_slots[student]:
550             slot = slots_copy[i]
551             course = random.choice(slot)
552             students[student].append(course)
553             filled_courses[i][course] += 1
554             if filled_courses[i][course] == 25:
555                 slots_copy[i].remove(course)
556
557     # 5 generate the timetable
558     al_timetable = {str(i): {str(j): {str(k): "G" for k in range(1, 11)}
559                             for j in range(1, 6)} for i in range(offset+1,
offset+501)}
560     for k, student in students.items():
561         for course in student:
562             for index, slot in enumerate(slots):
563                 if course in slot:
564                     day, p1, p2 = as_set_map[index][0].split("_")
565                     al_timetable[str(k)][day][p1] = rooms[course]
566                     al_timetable[str(k)][day][p2] = rooms[course]
567                     day, p1, p2 = as_set_map[index][1].split("_")
568                     al_timetable[str(k)][day][p1] = rooms[course]
569                     al_timetable[str(k)][day][p2] = rooms[course]
570                     day, p1, p2 = as_set_map[index][2].split("_")
571                     al_timetable[str(k)][day][p1] = rooms[course]
572                     al_timetable[str(k)][day][p2] = rooms[course]
573
574     timetable.update(al_timetable)
575     self.time_table.update(timetable)
576
577     return timetable
578
579 def add_pshe_gt(self):
580     """
581         Add PSHE session for each student.
582         For G1s and G2s, this lesson is added at P5 on everyday to immitate the 1-
period classes.
583     """
584     g1_offset = 23000
585     g2_offset = 22000
586     student_list = list(self.time_table.keys())
587     offset = 0
588     flag = True
589     for floor in self.avail_rooms.values():
590         for room in floor:
591             for i in range(offset, offset+25):
592                 if i >= len(student_list):
593                     flag = False
594                     break
595                 if int(student_list[i]) // 1000 == g1_offset // 1000 or
int(student_list[i]) // 1000 == g2_offset // 1000:
596                     for day in range(1, 6):
597                         self.time_table[student_list[i]][str(
598                             day)]['5'] = self.map_building(room) + str(room)
599                 else:
600                     gt_day = random.choice(["1", "2", "4", "5"])
601                     gt_floor = random.choice([6, 7, 8])
602                     gt_room = random.choice(

```

```

603             list(self.avail_rooms[gt_floor]))
604     if gt_day != "5":
605         self.time_table[student_list[i]][gt_day]['5'] =
606             gt_room) + str(gt_room)
607         self.time_table[student_list[i]][gt_day]['6'] =
608             gt_room) + str(gt_room)
609     else:
610         self.time_table[student_list[i]][gt_day]['7'] =
611             gt_room) + str(gt_room)
612         self.time_table[student_list[i]][gt_day]['8'] =
613             gt_room) + str(gt_room)
614     self.time_table[student_list[i]]['3']['5'] =
615             room) + str(room)
616     if not flag:
617         break
618     offset += 25
619     if not flag:
620         break
621
622     return self.time_table
623
624
625 if __name__ == "__main__":
626     # Need to generate multiple times so that the rooms not crash, therefore
627     # creating no errors
628     generator = TimetableGenerator()
629     time_table1 = generator.generate_g_level("G1")
630     time_table2 = generator.generate_g_level("G2")
631     time_table3 = generator.generate_as_al_level()
632     generator.add_pshe_gt()
633
634     with open("timetable_with_gt_0.json", "w") as file:
635         json.dump(generator.time_table, file, indent=4)

```