Pathfinding visualizer

Autorzy: Filip Gaweł, Piotr Białas

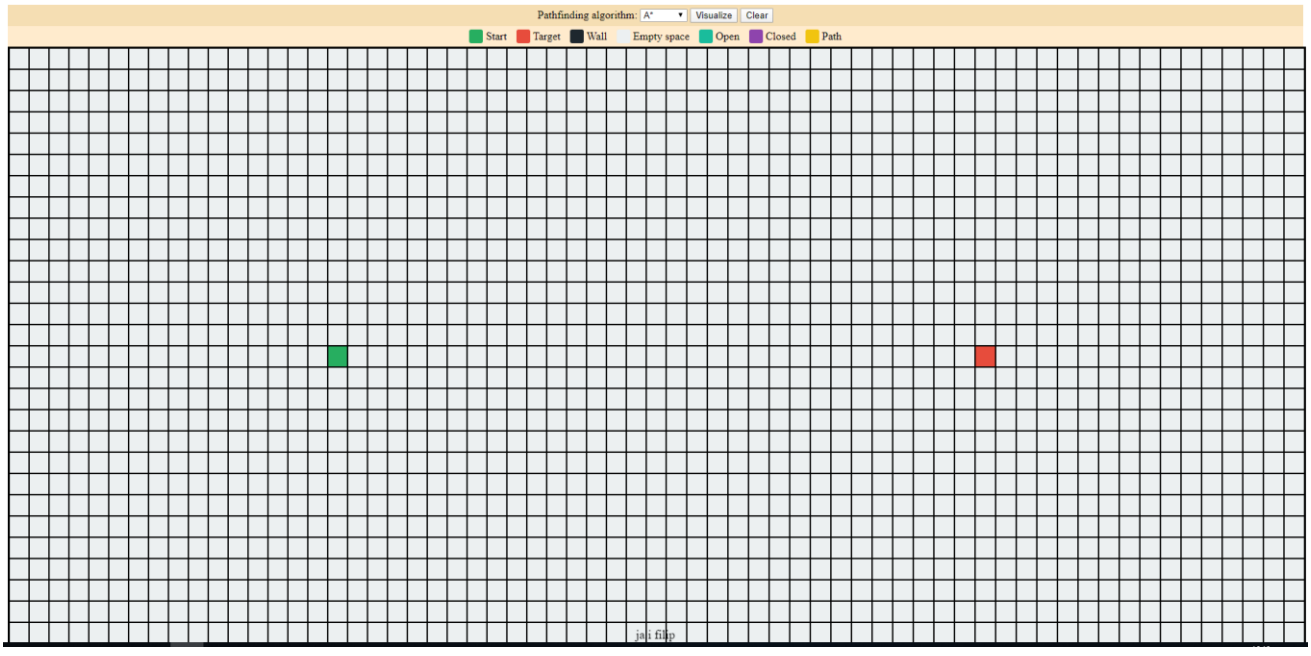
MS INF III Heurystyczne metody optymalizacyjne Gr. IA – Projekt V semestr

1. Część I

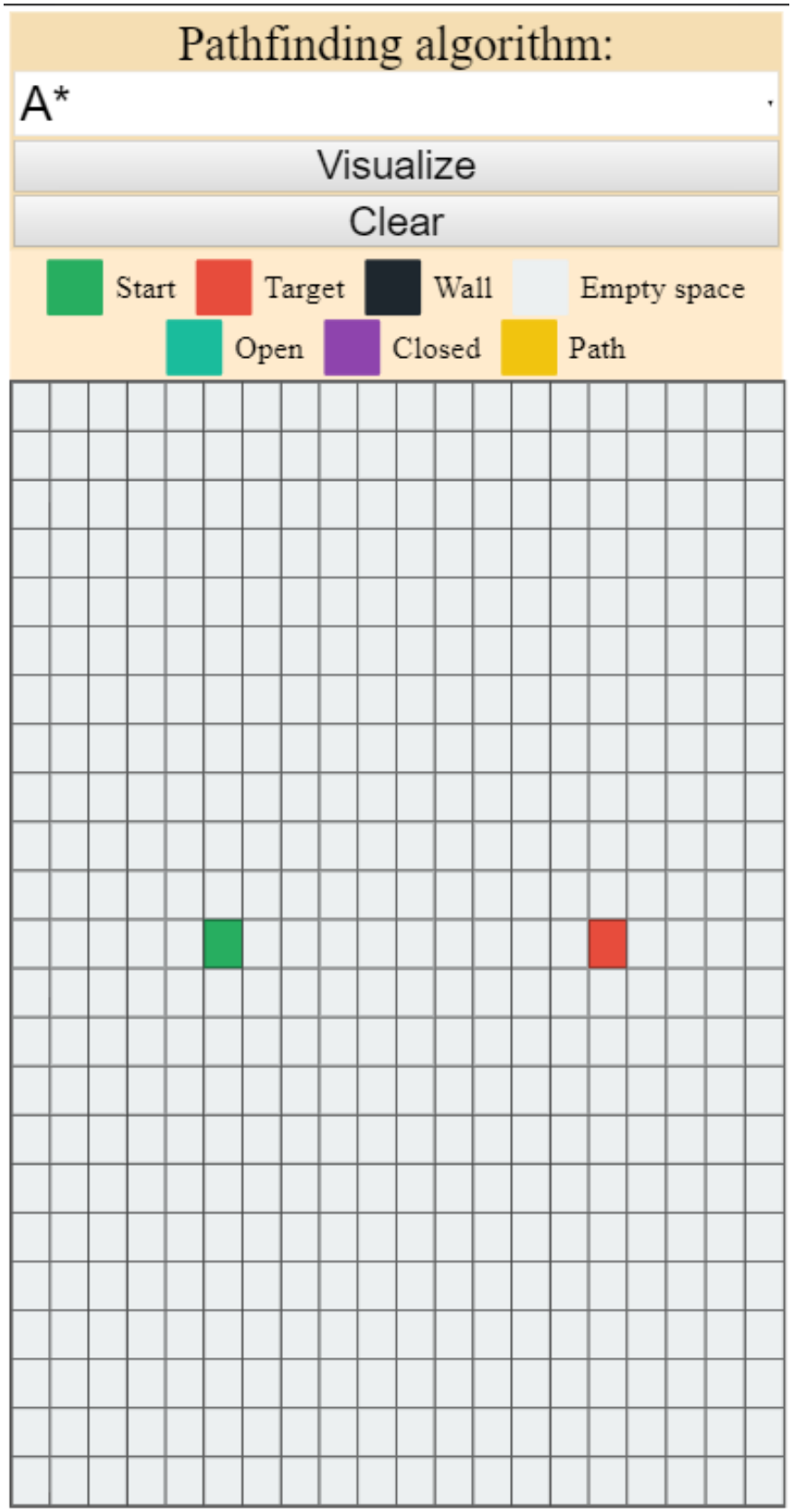
- **Opis programu**

Pathifinding visualizer jest aplikacją dostępną przez przeglądark. Na ekranie wyświetlane jest menu z wyborem algorytmu oraz siatka interaktywna. Program wyznacza najkrótszą drogę (za pomocą wybranego algorytmu), z punktu startowego (zielony kwadrat) do końcowego (czerwony kwadrat). Użytkownik poprzez klikanie lewym przyciskiem myszy może tworzyć ściany, które utrudnią wybranemu algorytmowi wyznaczenie drogi. Punkt startowy i docelowy może być przesuwany przez użytkownika poprzez kliknięcie myszą i przeniesienie w wybrane miejsce. Analogicznie użytkownik korzystający z urządzenia mobilnego może tworzyć ściany lub przesuwać skrajne punkty dotykając ekranu.

- **Wygląd głównego ekranu**



- **Wygląd na urządzeniu mobilnym**



2. Część II

- **Specyfikacja techniczna**

(a) Podział projektu na pliki:

- index.html *(69 lin)*
- node.js *(28 lin)*
- PriorityQueue.js *(58 lin)*
- script.js *(486 lin)*
- style_mobile.css *(135 lin)*
- styles.css *(139 lin)*

(b) Co znajduje się w danym pliku

node.js – klasa jednej komórki siatki

- Atrybuty:
  - x
  - y
  - type = “empty_cell”
  - gCost = 0
  - hCost = 0
  - parent = null
  - priority = Infinity
- fCost()
- Equals()

PriorityQueue.js – lista kolejkowa

- constructor()
- enqueue(element)
- dequeue()
- front()
- rear()
- isEmpty()
- refresh()

script.js – algorytmy oraz rysowanie

- init()
- CreateGrid()
- BuildWall()
- SetStartTarget()
- IdToNode()
- NodeToId()
- Distance()
- Neighbours(n)
- LowestFCost(nodes)
- AStar()
- Clear()
- DrawPath()
- ColorNode(node, type)
- Visualize()
- Dijkstra()
- resetNodes()
- CreateGridForPhones()
- dragStart()
- dragEnd()
- drag()
- slowDrawOpenPath()
- slowDrawOpenPathClose()

Index.html – podział strony

style_mobile.css – styl strony na urządzenia mobilne

styles.css – styl strony

- **Szczegóły techniczne**

a) Algorytmy/fragmenty kodu

- Do znalezienia najkrótszej ścieżki wykorzystaliśmy następujące algorytmy:
  - A* opisany przez Petera Harta, Nilsa Nilssona oraz Bertrama Raphaela.

```
201
202  function AStar() {
203      Clear("visualization");
204      let open = [];
205      let closed = [];
206      let start = grid[start_y][start_x];
207      let target = grid[target_y][target_x];
208
209      start.gCost = 0;
210      start.hCost = Distance(start, target);
211      open.push(start);
212
213      while (open.length > 0) {
214          let current = LowestFCost(open);
215          closed.push(current);
216          ColorNode(current, "closed");
217
218          let index = open.indexOf(current);
219          open.splice(index, 1);
220
221          if (current.Equals(target)) {
222              DrawPath(current);
223              //slowDrawOpenPathClose(open, current, closed);
224              return;
225          }
226
227          let neighbours = Neighbours(current);
228
229          neighbours.forEach(n => {
230              if (n.type != "wall" && closed.indexOf(n) < 0) {
231                  let temp = current.gCost + Distance(current, n);
232                  if (temp < n.gCost || open.indexOf(n) < 0) {
233                      n.gCost = temp;
234                      n.hCost = Distance(n, target);
235                      n.parent = current;
236
237                      if (open.indexOf(n) < 0) {
238                          open.push(n);
239                          ColorNode(n, "open");
240                      }
241                  }
242              }
243          });
244      }
245
246      alert("There is no path");
247      return null;
248  }
```

- Dijsktra opracowany przez holenderskiego informatyka Edsgera Dijkstrę

```
315  function Dijkstra() {
316      Clear("visualization");
317      let open = [];
318      let start = grid[start_y][start_x];
319      let target = grid[target_y][target_x];
320
321      resetNodes();
322      //odległość od źródła, wszystkie pozostale mają infinity
323      start.priority = 0;
324      let Q = new PriorityQueue();
325
326      open.push(start);
327
328      for (i = 0; i < height; i++) {
329          for (j = 0; j < width; j++) {
330              Q.enqueue(grid[i][j]);
331          }
332      }
333      while (!Q.isEmpty()) {
334          if (Q.front() == "No path!") return;
335          let minNode = Q.front();
336
337          open.push(minNode);
338
339          Q.dequeue();
340
341          if (minNode.Equals(target)) {
342              slowDrawOpenPath(open, minNode);
343              return;
344          }
345          let neighbours = Neighbours(minNode);
346          neighbours.forEach(n => {
347              if (minNode.priority + 1 < n.priority && n.type != "wall") {
348                  n.priority = minNode.priority + 1;
349                  //have to update priorityqueue
350                  Q.refresh(n);
351                  n.parent = minNode;
352              }
353          });
354      }
355      return null;
356  }
```