

PAPL Technical Report

Overview

I wanted to make a SMS/text like adventure, and I wanted to present this as SMS like app. I chose to do the project with an server-client architecture. I made the API with help of spring boot in java, and used react to make the frontend. the frontend will work in any browser thats supports ES6+.

Decision map design

My Decision Map was designed feel and look like SMS conversation, where a unkonen contact attempts to enlist your help to access your employer computer network with your help.

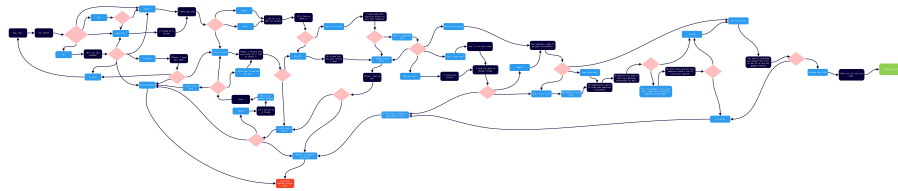


Figure 1: Decision Map Image

Changes to sample project and files

Given I have chosen to go with a server-client architecture webapp, I needed to make significant changes to original Project, thus I decided to create my own classes. I still used a linked-node map design.

Loading and building The decision map

Loading of the map was handled by the `loadDecisionMap()` method of the `DecisionMapController` class, this method is called after the api has started or on a HTTP GET request on path `/api/start`.

```
//api/src/main/java/up919313/PAPL_CW/DecisionMapController.java:31
@PostConstruct
private void loadDecisionMap(){
    JSONParser jsonParser = new JSONParser();
    ArrayList<JsonNode> nodes = new ArrayList<>();

    try (FileReader reader = new FileReader("newMap.json"))
    {
        //Read JSON file
        Object obj = jsonParser.parse(reader);
    }
```

```

        JSONArray jsonArrayNodes = (JSONArray) obj;
        jsonArrayNodes.forEach(n -> {
            JsonNode node = new JsonNode((JSONObject) n);
            nodes.add(node);
        });
        this.decisionMap = new DecisionMap(nodes.toArray(JsonNode[]::new));

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

For my project when with JSON to save the Decision Map as JSON is easier to handle than csv files. This meant I needed to load and parse JSON into the Java backend. I was able to achieve this with use of `FileReader` and `JSONParser` from `org.json.simple.parser`. The logic was quite simple, load in the file as a `Object` from the `FileReader` then cast it as a `JSONArray` so I can then loop through each `JSONObject` in the `JSONArray` and create a new instance of custom class `JsonNode` which was appended to `ArrayList<JsonNode> nodes`. In the constructor for `JsonNode` I parsed the json object to a java object.

```

// api/src/main/java/up919313/PAPL_CW/JsonNode.java:25
public JsonNode(JSONObject node){
    this.node_id = Math.toIntExact((Long) node.get("node_id"));
    this.node_text = (String) node.get("node_text");
    this.node_type = getNodeFromTypeString((String) node.get("node_type"));
    JSONArray JSONlinkedNodeIds = (JSONArray) node.get("linked_nodes");
    this.linked_nodes = new int[JSONlinkedNodeIds.size()];
    JSONlinkedNodeIds.forEach(id -> this.linked_nodes[JSONlinkedNodeIds.indexOf(id)] = Math.toI
}

```

I was then able to assign a new instance of `DecisionMap` to `DecisionMapController.decisionMap` field, after passing in the `ArrayList<JsonNode> nodes`. The constructor of `DecisionMap` takes a map in the form of a `JsonNode[]`. Before I can build the links I need to build each `DecisionNode`, this is done by the static `DecisionMap.buildDecisionNodes()`. This method returns an `DecisionNode[]`. I'm then able to build the links by passing the array from `DecisionMap.buildDecisionNodes()` into `DecisionMap.buildDecisionMap()`, I set `DecisionMap.head` to be the node with an ID of 0. I then call `DecisionNode.buildDecisionNode()` on the head. In `DecisionNode.buildDecisionNode()` made and populated arrays containing a reference to a linked node, this makes for easy traversal of the map.

```

// api/src/main/java/up919313/PAPL_CW/DecisionNode.java:38
public void buildDecisionNode(DecisionNode[] decisionNodes){
    try {

```

```

        this.linkedNodes = new DecisionNode[linkedNodeIds.length];
        for (int i = 0; i < this.linkedNodeIds.length; i++) {
            this.linkedNodes[i] = DecisionMap.findNodeInArray(linkedNodeIds[i], decisionNodes);
            // System.out.println(this.toString());
        }
        for (DecisionNode linkedNode : this.linkedNodes) {
            if (linkedNode.linkedNodes == null){
                linkedNode.buildDecisionNode(decisionNodes);
            }
        }
    }
    catch (NodeNotFoundException e){
        System.out.println(e.getMessage());
    }
}

```

Handling API requests

As mentioned above there is a single instance of the decision map, this means a request to traverse the map will change the head node. this behaviour is useful as the server to keep track of the head and handle all traversal logic, however this does have some drawbacks which include not being able to support multiple instances of the game.

a game is started by a request to `/api/start` this will load the decision app again, this will ensure that the player gets to start the game from the beginning. to traverse to the next node the client will send a request to `/api/next/{nodeID}` where `nodeID` is the next node. this will trigger the `DecisionMapController.next()` method which will then call `DecisionMap.pickNode()`. This simple method will loop through the currentNode's `LinkedNodes`, and will set the `DecisionMap` head to the linked node that matches the `nodeID` given in the request path, and will then return the node to client.

```

//api/src/main/java/up919313/PAPL_CW/DecisionMap.java:18
public DecisionNode pickNode(int nodeId){
    DecisionNode currentNode = this.getHead();
    for (DecisionNode node: currentNode.getLinkedNodes()) {
        if(node.getNodeId() == nodeId){
            this.head = node;
            return node;
        }
    }
    return null;
}

```

Design and usa of custom exception handlers

My use of custom exception handlers is somewhats lacking due time constraints and that fact that most error and error check happands in the Webapp. that said I did make a `NodeNotFoundException` that is thrown when a node can not be found.

Third party and open source libraries/software used

- API(Java)
 - json-simple (<https://code.google.com/archive/p/json-simple/>)
 - Spring Boot (<https://spring.io/projects/spring-boot>)
 - Spring Web (<https://mvnrepository.com/artifact/org.springframework/spring-web>)
- WebApp
 - ReactJS (<https://reactjs.org/>)
 - react-router-dom (<https://github.com/ReactTraining/react-router/blob/master/packages/react-router-dom>)
 - autoscroll-react (<https://github.com/thk2b/autoscroll-react>)
 - tailwindcss (<https://tailwindcss.com/>)
 - Font Awesome (<https://fontawesome.com/>)
- Reverse Proxy
 - Caddy (<https://caddyserver.com/>)

Video Link

<https://youtu.be/2bC2JrUWHZM>