Documentation

- Backend description
- VFG File description
- Developer Information Frontend

Backend description

Backend provides Restful API to parse pddl files and return VFG file.

The API is implemented in python using Django and Django rest_framework

Request description

To send data to the server you must:

- make a POST request must be directed to http://hostAddress/upload/pddl
- Content-Type must be multipart/form-data
- Data must be passed in body argument.
- It should provide 'domain', 'problem' and 'animation' keys, and respective files as values.
- 'url' key is optional and will change the url for a solver
- the values for each document to send it as string

https://immense-bastion-42146.herokuapp.com/upload/pddl

API has returned a VFG file.

Requet example in Javascript

Javascript snippet example to show the form of a POST request to the server.

fetch

```
<body>
    <h1>Please open a file</h1>
    <p>Domain</p>
    <input type="file" id="domain" multiple>
    <p>Problem</p>
    <input type="file" name="prob" id="problem">
    <p>Animation</p>
    <input type="file" name="anim" id="animation">
    <input type="button" value="submit" id="send">

    <div id="txt"></div>=
    <script>
        const upload = (file) => {
            const formData = new FormData();
            for(const name in file){
                formData.append(name, data[name]);
            }
            let resp;

            fetch('http://127.0.0.1:8000/upload/pddl', { // Your POST endpoint
                method: 'POST',
                body: formData
                }).then(
                    response => resp = response.json()
                ).then(
                    success => console.log(success)
                ).catch(
                    error => console.log(error)
                );
            return resp;
        }

        let texts = document.getElementById('txt');
        const domainIn = document.getElementById('domain');
        const problemIn = document.getElementById('problem');
        const animationIn = document.getElementById('animation');
        const submit = document.getElementById('send');
        let data = {};

        domainIn.addEventListener('change', (event) => {
            event.target.files[0].text().then(result=> data.domain = result);
        });
        problemIn.addEventListener('change', (event) => {
            event.target.files[0].text().then(result=> data.problem = result);
            //data.problem = event.target.files[0].text();
        });
        animationIn.addEventListener('change', (event) => {
            event.target.files[0].text().then(result=> data.animation = result);
            //data.animation = event.target.files[0].text();
        });

        submit.addEventListener('click', (event) => {
            let resp = upload(data);
            console.log(resp);
            texts.innerText = resp;
        });
    </script>
</body>
```

VFG File description

VFG file is an JSON file containing information for the visualisation.

At its root:

1) visualStages: It contains a list of objects to describe a stage. The length of the list corresponde to the total number of stages.

Every object has:

visualSprites: A list of sprite elements and its atributes, including type, color and position.

StageName: Action performed.

Stage Info: Snipet of pddl code for the stage.

isFinal: Boolen indicating if the current stage is the goal state.

2) subGoalPool: It contains an object with a key list and a value list.

3) subGoalType: It contains an object with a key list and a value list.

4) imageTable: It contains an object with a key list and a value list.

Every key correspond to a type of sprite, and every value has a code base64 PNG image.

To process a base64 image code, we need to use the code to create a URL into a <img> component such as:

```
<img src="data:image/png;base64,
iVBORw0KGgoAAAANSUhEUgAAAEoAAABUCAYAAAAlDKGaAAABdklEQVR4Xu3cPU7DQBhF0cnKYGfAzsLO0Efl
/IBuEzQox1WKJySOb1yNc1quJHBKK6O1G9TLWuv1cF/Oa63PHe7TblBva633A8x8
/gB1KwAqVgEKVBSIM0WBigJxpihQUSDOFAUqCsSZokBFgThTFKgoEGeKAhUF4kxRoKJAnCkKVBSIM0WBigJxpihQUSDOFAUqCsSZokBFgThTFKgo
EGeKAhUF4kxRoKJAnCkKVBSIs62LmlO4x5O48X96yGzrU8HXd/EhAv/9j87x6TmiPFiuXwQGSlEhkXtFzZsC88bAs1
/z3J5n5vd1D2reFDi+PfCsYBePJFA
/ZwAqfkVAgYoCcaYoUFEgzhQFKgrEmaJARYE4UxSoKBBnigIVBeJMUaCiQJwpClQUiDNFgYoCcaYoUFEgzhQFKgrEmaJARYE4UxSoKBBnigIVBeJ
MUaCiQJwpClQUiDNFgYoCcXZT1PWbC3Mi2Kngq/P3u/2eebzZfzK7KOoLP13HqksMK+sAAAAASUVORK5CYII" />
```

That component can be passed to a PIXI sprite. Then, the sprite adquire additional attributes.

```
const image = document.querySelector('img')
const base = new PIXI.BaseTexture(image);
const texture = new PIXI.Texture(base);
const sprite = new PIXI.Sprite(texture);
sprite.anchor.x = 0.5;
sprite.anchor.y = 0.5;
sprite.position.x = 50;
sprite.position.y = 50;
```

to wrap this behavior in a single element the class component Base64ToSprite can handle it.

5) message: Empty line.

Developer Information Frontend

The Visualiser takes a Visualisation File and displays an animation to the user. It is a ReactJS web app that renders animation objects with PixiJS.

Modifying the visualiser is only required when the current visualiser cannot adequately render the domain on-screen. It does not concern the logic of object layout. For example, a user might want to extend the Visualiser to support animated sprites.

Extending the Visualiser requires building the project in NodeJS. Modifications or extensions of the Visualiser need only be carried out on React only. For information on the Visualiser, see the file 'Visualiser Documentation'.

1. Getting started

Installation

See Deployment

Usage

See the User Manual, which is accessible from within the application, or with the file https://planimation.github.io/documentation/ug1/

Development

The visualizer will takes the visualization file (VFG) generated by the visualization file generator. The visualizer will automatically analyze the visualization file and transfer the JSON data into a PixiJS App Scene and Sprites and display them on the screen.

Editing Sprites

The screen component is the file that contains the code to render the Sprites elements. VFg were original designed to run in Unity Engine for WebGL, therefore the axis origin is set at the bottom left, while in Javascript the defult origin is top left. A wrapper function { } translates this into the proper coordinates and creates a new sprite in the screen. To add more properties the wrapper function needs to be updated to parse additional changes and map them to sprite objects. To kno more about sprite properties check: https://pixijs.download/dev/docs/PIXI.Sprite.html

Additionally the function difference in pagaFour/index.js need to be updated to handle animated transition to this new properties.

How to Run

To check the changes, React provides a developement server that updates. Simply run:

```
npm start
```

Building

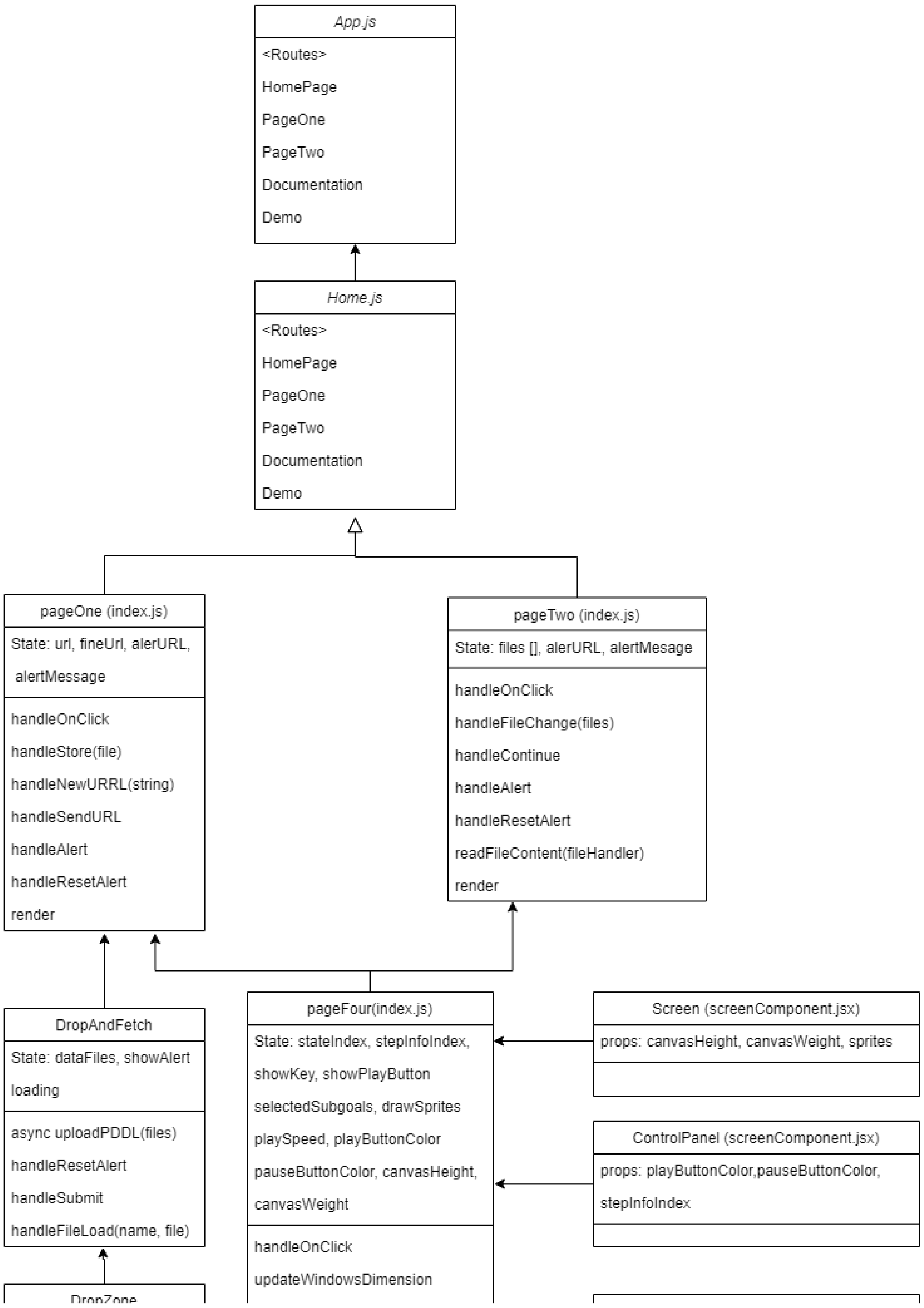
To build the application simply run the command:

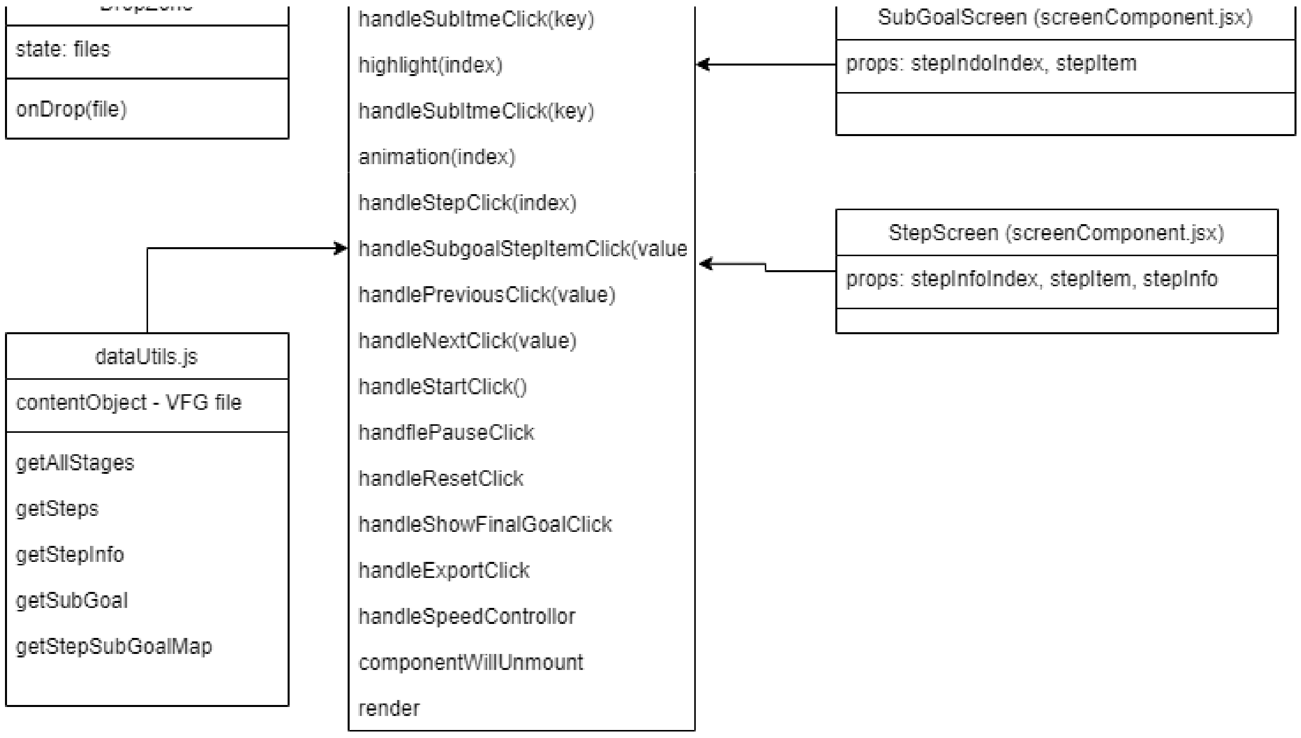
```
npm build
```

2. Architecture Overview

Input: a visualisation file. The visualisation file is a generic JSON file generated by the VFG, demonstrating how shapes are to be drawn on the screen in stages. The structure of the Visualisation File is outlined in the VFG Documentation file. Output: a visualisation of the visualisation file

Component structure design


App.js
<Routes>
HomePage
PageOne
PageTwo
Documentation
Demo
Home.js
<Routes>
HomePage
PageOne
PageTwo
Documentation
Demo
pageOne (index.js)
State: url, fineUrl, alerURL,
alertMessage
handleOnClick
handleStore(file)
handleNewURRL(string)
handleSendURL
handleAlert
handleResetAlert
render
pageTwo (index.js)
State: files [], alerURL, alertMesage
handleOnClick
handleFileChange(files)
handleContinue
handleAlert
handleResetAlert
readFileContent(fileHandler)
render
DropAndFetch
State: dataFiles, showAlert
loading
async uploadPDDL(files)
handleResetAlert
handleSubmit
handleFileLoad(name, file)
DropZone
pageFour(index.js)
State: stateIndex, stepInfoIndex,
showKey, showPlayButton
selectedSubgoals, drawSprites
playSpeed, playButtonColor
pauseButtonColor, canvasHeight,
canvasWeight
handleOnClick
updateWindowsDimension
Screen (screenComponent.jsx)
props: canvasHeight, canvasWeight, sprites
ControlPanel (screenComponent.jsx)
props: playButtonColor,pauseButtonColor,
stepInfoIndex



Directory Structure

```
frontend-js/
 config/
 cypress/
  fixtures/
   demoData/
  integration/
   examples/
   Student.spec.js
  plugins/
  support/
 docs/
 prototypes/
 public/
 scripts/
 src/
  components/
   navigationBar/
    navigationBar.js
    navigationBar.module.less
   sample/
    Sample.js
    Sample.module.less
   Template/
    index.js
    index.module.less
   alertInFormat.jsx
  pages/
   HomePage/
    home.js
    index.js
    index.module.less
   PageFour/
    dataUtils.js
    index.js
    index.less
   PageOne/
    dropAndFetch.jsx
    dropAndFetch.test.js
    dropZone.jsx
    dropZone.test.js
    index.js
   PageThree/
    index.js
   PageTwo/
     index.js
     pageTwo.test.js
  plugin/
   plugin.js
  Styles/
   index.module.less
  tests/
   App.test.js
   readme.md
  App.js
  App.module.less
  index.css
  index.js
  reportWebVitals.js
  setupTests.js
 .dockerignore
 .gitignore
 .travis.yml
 cypress.json
 Dockerfile
 jsconfig.json
 nginx.conf
 package-lock.json
 package.json
 ReadMe.md
 static.json
 yarn.lock
```

3. Components

Components are structure via a number of subdirectories to the Visualiser folder

Pages

HomePage: Landing page, shows four link to access pageOne, pageTwo, Documentation and demo video

PageOne: shows the Build visualisation from problem user interface

PageTwo: shows the Build visualisation from VFG user interface

PageFour: shows the visualisation screen user interface

Function components

Auxiliary component to privide pages functionality.

dataUtils

Constains functions to parse the VFG file and extract textures.

Style components

css files and Material-UI theme configurable in home.js

Testing

It is divided in two folders:

- cypress: contains the end-2-end automated testing an testing files.
- Test: contains unit and integration automated test files in jest:

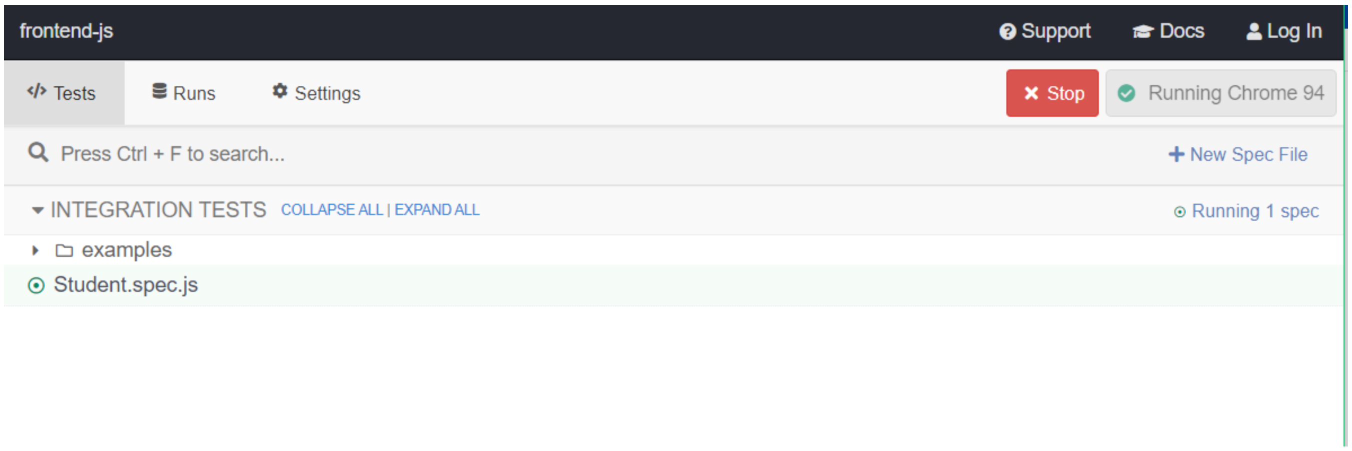
To run unit and integration test run:

> *npm test*

to run end-2-end automated test, run

> *npx cypress open*

Wait for the panel to open and select file Student.spec.js to run student user cases automated test.



4. Extensions

The visualizer is designed as a component that only displays data in the visualization file.

Most extensions to Planning Visualiser can be made through the VFG. If new types of animiations need to be supported which cannot be represented in the existing Visualsiation File format: * Define a new object property in the visualization file