Hassan Badru - Jun 2018

Decode Vector Instruction Strings - Art+Logic Exercise

Description

Using the decoding function written for Part 1 submission to decode and describe a set of simple commands to build a vector-based drawing system.

Set Up / Running App

- Download & unpack Hassan_Badru_Part2.zip file
- (*Recommended*) Install & run a Virtual Environment (e.g. `source Art_Logic_Env/bin/activate`)
- Assuming you already have *python* and *pip*, install requirements using `pip install -r requirements.txt` within command prompt or terminal *(if not, check out How to install python & pip)*
- Assuming you already have *Node* or *NPM*, inside the **frontend** folder, install node_modules (dependencies) using `npm install` *(if not, check out How to install Node)*
- To start server, run the command `python manage.py runserver`
- On your browser, go to http://127.0.0.1:8000/ or local server address provided within terminal

Task

This system uses a pen-based model where an imaginary pen can be raised or lowered. The commands supported in this mini-language are:

- Clear the drawing and reset all parameters
- Raise/lower the pen
- Change the current color
- Move the pen

Commands are represented in the data stream by a single (un-encoded) opcode byte that can be identified by always having its most significant bit set, followed by zero or more bytes containing encoded data values.

Any unrecognized commands encountered in an input stream should be ignored.

Commands

Here are command formats used

CLEAR	
Command	CLR
Opcode	FO
Parameters	(None)
Output	CLR;\n

PEN	
Command	PEN

Opcode	9O
Parameters	0=up,not 0=down
Output	PEN UP or PEN DOWN

COLOR	
Command	CO
Opcode	AO
Parameters	R G B A
Output	CO {r} {g} {b} {a}

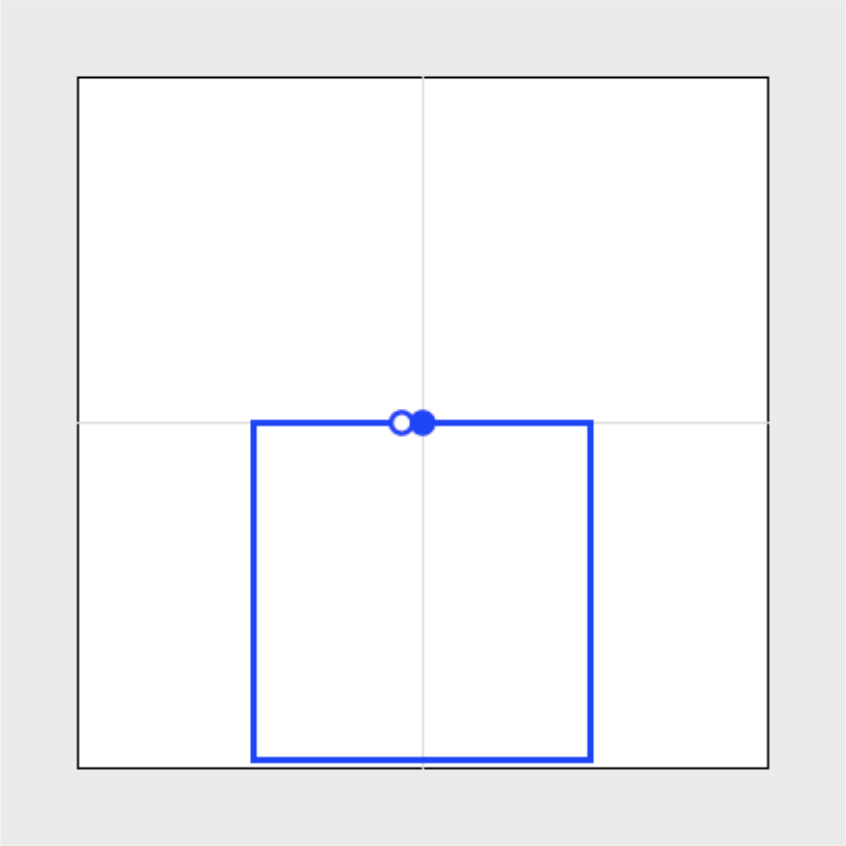
MOVE	
Command	MV
Opcode	CO
Parameters	dx0 dy0 [dx1 dy1 .. dxn dyn]
Output	MV (xo, y0) (x1, y1) [... (xn, yn)]

Example

Blue Square:

Input Data:

```
F0A040004000417F417FC0400040009040004768 4F5057384000804001C0
5F2040004000014001404000400 07E405B2C4000804000
```



Output:

```
CLR;
CO 0 0 255 255;
MV (0, 0);
PEN DOWN;
MV (4000, 0) (4000, -8000) (-4000, -8000) (-4000, 0) (-500, 0);
PEN UP;
```

Technology Stack Used

- **HTML5 / CSS (View Template)**

- **REACT JS (Frontend)**
- **Django / Python (Backend)**
- **PostgreSQL (Database)**
- **Django REST Framework (API)**
- **Node / NPM (Production Build)**

Structure

- ### Model

```
class UserAction(models.Model):
    operation = models.CharField(max_length=100, default='encoding')
    input = models.CharField(max_length=100, default='8191')
    result = models.CharField(max_length=100, default='')
```

- ### View

```
class ArtLogicAPI(generics.ListCreateAPIView):
    queryset = UserAction.objects.all()
    serializer_class = UserActionSerializer
```

```
class ArtLogicApp(TemplateView):
    template_name = 'index.html'
```

- ## Template

Using ReactJS as frontend

I set the template use to load from React's production build folder in *settings.py* `'DIRS': [ os.path.join(BASE_DIR, 'frontend/build') ],`

Node Dependencies (Package.JSON)

```
"react": "^16.3.2",
"react-dom": "^16.3.2",
"react-scripts": "1.1.4",
"react-vis": "^1.9.4"
```

One External Library used (react-vis): For creating X-Y plot of coordinates on canvas

- ## Utility (MyFunction.py)

Decoding function:

Takes in 16-bit hexadecimal

```
def decoder(input_num):
    ....
```

```
    return output
```

- returns a decoded decimal within the range of [-8192, 8191]

Get function:

Searches the inputted string for all commands present within, using regular expressions.

```
def get_instructions(m_string):
    ....
    return output
```

- return a stored dictionary of every command within the string, with their respective starting positions as keys

Read function:

Takes in the result from `get_instruction()`, creates an action log of all possible parameters and then decodes them

```
def readInstruction(s1):
    ....
    return output
```

- returns an array of objects that includes all instruction as either strings or keys for accompanying parameters, sorted in order of appearance

Boundary Fix function:

Checks if value exceeds boundary,

```
def def fix_boundary(val):
    ....
    return output
```

- returns value if less or border if more

Write function:

Takes instruction stream from `readInstruction()` and adds pen logic by performing operations from all instructions on their parameters (i.e. clear, change color, draw). For easy conversion into JSON, creates a dictionary for all results, using order number as keys and entire output as values. The function is able to compute right boundary coordinates for pen when either x and/or y are outside or returning into boundary: [-8192, 8191].

```
def write_instructions(instruction_stream):
    ....
    return output
```

- returns a JSON object that can be understood by Javascript/ ES6is and then passed on to the frontend as a variable

- ## Routing

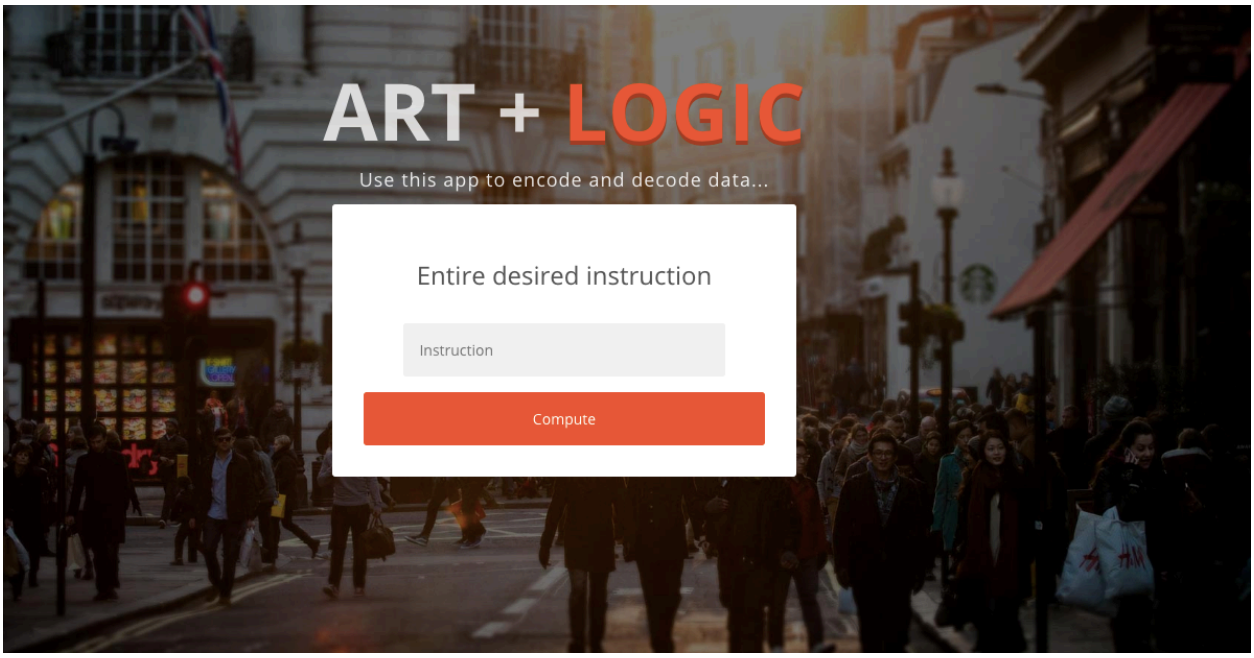
Django App (art_logic_app):

```
    url('api/', views.ArtLogicAPI.as_view() ),
    url(r'^$', views.ArtLogicApp.as_view(),
name="art_logic" )
```

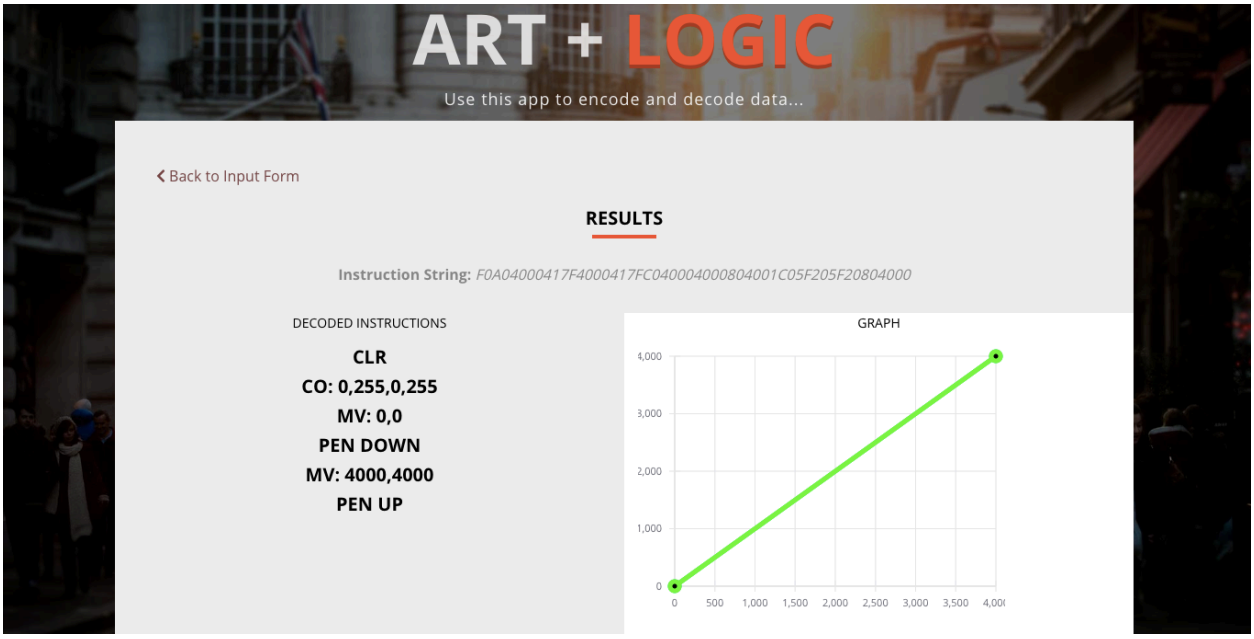
How the App Works

REACT Single Page

Input Instruction Screen:



Decoded Instruction Screen & Graph:



Features

- User can input a un-encoded string that contains the instructions they want to execute with hexadecimal parameters

- The app takes in the string and computes result if valid

- Movements are only drawn when the pen is down. The app displays all movement instructions with current positions while following the pen's logic

- Result are displayed in two forms:
  Textual: Shows all commands and effects of pen positions on new coordinates in form of an unordered HTML list `<ul>`. User can scroll down to see more of longer results.

  Graphical(*additional*): Shows the points drawn by the commands as pen if forced to stay stay within boundary. However, the pen always draws as it moves along (regardless of pen positions) due to UI design time constraint.

- Error Handling: The program tries hard to translate every piece of information from strings in recieves and displays what it understands. If any instruction exists, the app displays it, else users simply won't receive any output on result display screen. If there is no input, user won't leave input screen.

- The app was designed to ignore unrecognized .

- The App plots and styles the graph once it reads and understands the decoded instruction correctly.

Folder Structure

```
Hassan_Badru_Part2
    ├── README.md
    ├── art_logic
    │   ├── __init__.py
    │   ├── settings.py
    │   ├── urls.py
    │   ├── wsgi.py
    ├── art_logic_app
    │   ├── __init__.py
    │   ├── admin.py
    │   ├── apps.py
    │   ├── fixtures
    │   │   └── art_logic_app.json
    │   ├── migrations
    │   │   ├── 0001_initial.py
    │   │   ├── 0002_auto_20180515_2039.py
    │   │   ├── 0003_auto_20180519_2226.py
    │   │   ├── __init__.py
    │   │   └── __init__.pyc
    │   ├── models.py
    │   ├── myfunctions.py
    │   ├── serializers.py
    │   ├── tests.py
    │   ├── urls.py
```

```
    │   ├── views.py
    ├── db.sqlite3
    ├── frontend
    │   ├── README.md
    │   ├── build
    │   │   ├── asset-manifest.json
    │   │   ├── favicon.ico
    │   │   ├── index.html
    │   │   ├── manifest.json
    │   │   ├── service-worker.js
    │   │   └── static
    │   │       ├── css
    │   │       │   ├── main.a203576f.css
    │   │       │   └── main.a203576f.css.map
    │   │       ├── js
    │   │       │   ├── main.c96fb44e.js
    │   │       │   └── main.c96fb44e.js.map
    │   │       └── media
    │   │           └── intro-bg.fb30f247.jpg
    │   ├──
    │   ├── package.json
    │   ├── public
    │   │   ├── favicon.ico
    │   │   ├── index.html
    │   │   └── manifest.json
    │   ├── src
    │   │   ├── App.js
    │   │   ├── App.test.js
    │   │   ├── css
    │   │   │   ├── App.css
    │   │   │   └── index.css
    │   │   ├── img
    │   │   │   └── intro-bg.jpg
    │   │   ├── index.js
```

```
    │   │   ├── logo.svg
    │   │   └── registerServiceWorker.js
    │   └── yarn.lock
    ├── requirements.txt
    ├── manage.py
    └── media
        └── ConvertedData.txt
```

Scalability

- I built this app in a way that allows me or any developer scale and integration past implementations with any possible future solutions.

Extensibility

- In the future, the graphical UI feature could be upgraded to more advanced SVG-based frameworks to better depict the pen logic which has been represented in the text output.
    - Draw line only when pen is down
    - Differentiate data marks when pen is up (donut) vs when pen is down (solid)
- Generate specific messages for user to learn why any string without an output only exists outside the scope of this decoding language.