**Project Name:** Visualization and Behavioural Analysis on Bayesian Hierarchical Inverse Reinforcement Learning (BHIRL) for the Toy Car IRL Task.

**Requirements stated by the client:**

**Task 1)**

1. Saving of trained (IRL) skills & adding feedback from the user (not ok, can’t tell, ok) on trained skills.

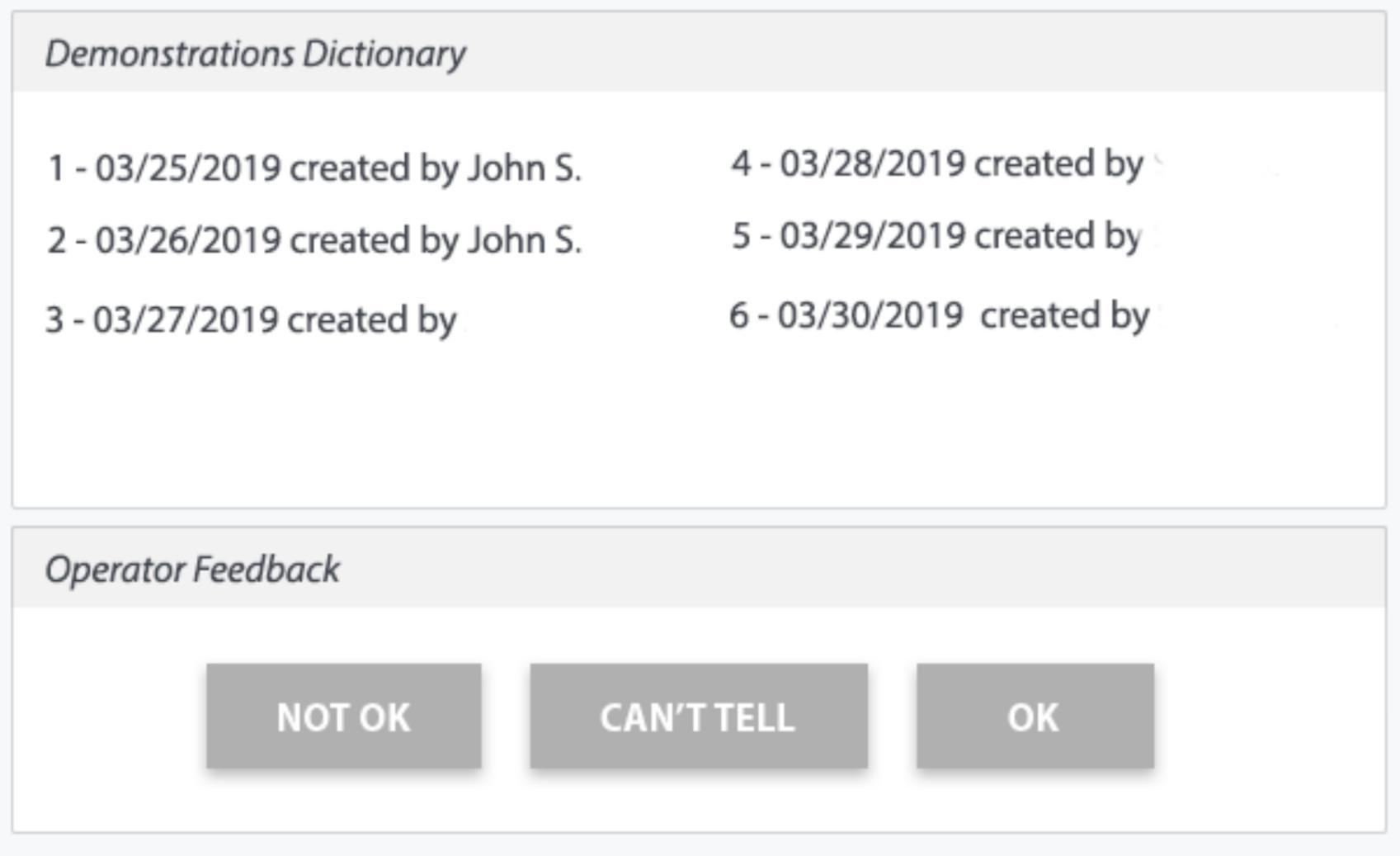


Fig 1. UI Screen

Operator feedback allows simple Collaboration with users. Integration of user feedback.

Question:

A) saving the trained skills, means you want to save the weights of the neural network to predict the next action for a particular behaviour?

Yes, We can add a feedback option at the end of the playing of the toy car using trained weights.

Answer:

I want to have the ability to present a skill, let say a few times and I want to save those presentations. Then (it’s done by the original script == Apprenticeship Learning via IRL) I want to optimize one skill but based on the amount of added presentations/demonstrations. So every time I add a new particular skill presentation/demonstration, I want to be able to save them and optimize a particular skill.

Yes, feedback can be added. This should help learn faster, more efficiently.

Question:

So, you require a UI for presenting an ability to trigger a skill (for eg. red behaviour) which would be played using a previous trained model for that particular skill. Then, at the end of the demonstration of that skill(for eg. red behaviour) we need to showcase the UI having the above Fig. 1 which shows the list of demonstrations and a block below which has buttons to take the feedback from the user (operator) who is performing the demonstration. Then this feedback needs to be stored somewhere (where exactly?) from where it can be accessed later. Is this correct?

Based on the feedback you want to track which skill demonstration has reached upto the mark performance and you need not focus on this skill any more. We can move to demo other skills now. (Is this the aim?)

Answer:

Regarding UI, yes I would suggest to do it in Jupyter Notebook, that’s what other developers do (I can send a sample visualization code if that helps). In case we might find challenging implementing ‘buttons, that’s very much okay. We can visualize as much as we can and in Jupyter type e.g. 1 for “NOT OK”, 2 for “CAN’T TELL” etc.

*Re Then this feedback needs to be stored somewhere (where exactly?) from where it can be accessed later. Is this correct?*

Yes, I want to store the feedback. XML file is fine as it would be very easy for you to implement. What do you propose? Someone proposed ScyllaDB or graph DB - Neo4j both sounds interesting. See: <https://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis> and [http://nosql-database.org](http://nosql-database.org/) but simple solution might okay enough for now.

*Re Based on the feedback you want to track which skill demonstration has reached upto the mark performance and you need not focus on this skill any more. We can move to demo other skills now. (Is this the aim?)*

Let’s say, you and your colleague make one skill demonstration (for eg. red behaviour) for an agent, we all need to know who made a specific demonstration hence the names along the demonstrations. Let’s say, I know how the agent should exactly behave or what I don’t like in the agent so I want to give you both feedback (not ok, can’t tell, ok).

Based on that feedback, (it’s done by the original script == Apprenticeship Learning via IRL) I want to (re)optimize one skill (for eg. red behaviour) based on the amount of added skill demonstrations. So every time you or your colleague add a new particular skill demonstration, we want to be able to save them, give feedback, and (re)optimize a particular skill (for eg. red behaviour).

Note: **(We created yml file using : conda create -n BHIRL3 python=3.6 pip numpy keras tensorflow)**

**Steps to be followed at your end:**

**Approach 1: (Easier)**

In case you want to create a separate virtual environment, which doesn’t conflict with any of your existing library dependencies. **(Recommended)**

1. Download the zip folder () and extract in your local machine.
2. Move to the extracted folder from the terminal.
3. Open terminal and run the below command:

**sudo bash main.sh**

**(This runs th GUi from main.py, but if you want to run in jupyter, close this UI and follow the below steps:**

**OR** run the main.ipynb jupyter notebook:

* Open terminal and type below command to open jupyter:
* jupyter notebook
* Run the first cell using (Shift+Enter) and it will launch the GUI**)**

1. Type the User’s Name and select the behaviour and epochs and frames from dropdowns having available trained skills in the folders.
2. Press “Run” button to play the skill. Based on the demonstration, the user can give the feedback from the feedback buttons below and it will show the details of the demo in the Demo Dictionary above.

(**Note**:- Here the agent will start after approx 20-30 seconds due to iterations for learning bayesian probabilities)

1. where, in the above example:- Behavior/Skill = Red, Iteration = 2, Frames = 3000. Using these it will call the weight files which are stored in behavior folder respectively. Check the weight saved in the respective behavior folders.

(**Before you deactivate you can test Task2A by running commands as given on Page 13 of the document**)

1. Deactivate the virtual environment by typing the below command in terminal:

conda deactivate

**Approach 2:(Complex)**

In case you want to create a separate virtual environment, which doesn’t conflict with any of your existing library dependencies. **(Recommended).**

1. Download the zip folder () and extract in your local machine.
2. The extracted folder will have env.yml.
3. Move to the extracted folder from the terminal
4. **Install Pygame's dependencies with:**

* **sudo apt install mercurial libfreetype6-dev libsdl-dev libsdl-image1.2-dev libsdl-ttf2.0-dev libsmpeg-dev libportmidi-dev libavformat-dev libsdl-mixer1.2-dev libswscale-dev libjpeg-dev**

1. Then create a virtual environment by typing the below command and later install pymunk and pygame in root.

* **conda env create -f env.yml**

(**Note**: “BHIRL3” is the name of the virtual environment.)

Activate virtual environment by typing the below command in terminal:

* **conda activate BHIRL3** (All the dependencies will be installed)

**Install Pymunk separately:**

* **cd pymunk-pymunk-4.0.0**
* **python3 setup.py install**

Now again navigate to the repository with the toycar code

* **cd ..**

1. Run the evaluating code by typing the below command in terminal:

* **python3 main.py**

**OR** run the main.ipynb jupyter notebook:

* Open terminal and type below command to open jupyter:
* jupyter notebook
* Run the first cell using (Shift+Enter) and it will launch the GUI
* Type the User’s Name and select the behaviour and epochs and frames from dropdowns having available trained skills in the folders.
* Press “Run” button to play the skill. Based on the demonstration, the user can give the feedback from the feedback buttons below and it will show the details of the demo in the Demo Dictionary above.

(**Note**:- Here the agent will start after approx 20-30 seconds due to iterations for learning bayesian probabilities)

* where, in the above example:- Behavior/Skill = Red, Iteration = 2, Frames = 3000. Using these it will call the weight files which are stored in behavior folder respectively. Check the weight saved in the respective behavior folders. They are stored using the following conventions:-

(**Before you deactivate you can test Task2A by running commands as given on Page 13 of the document**)

1. Deactivate the virtual environment by typing the below command in terminal:

* conda deactivate

1. Visualization Techniques:

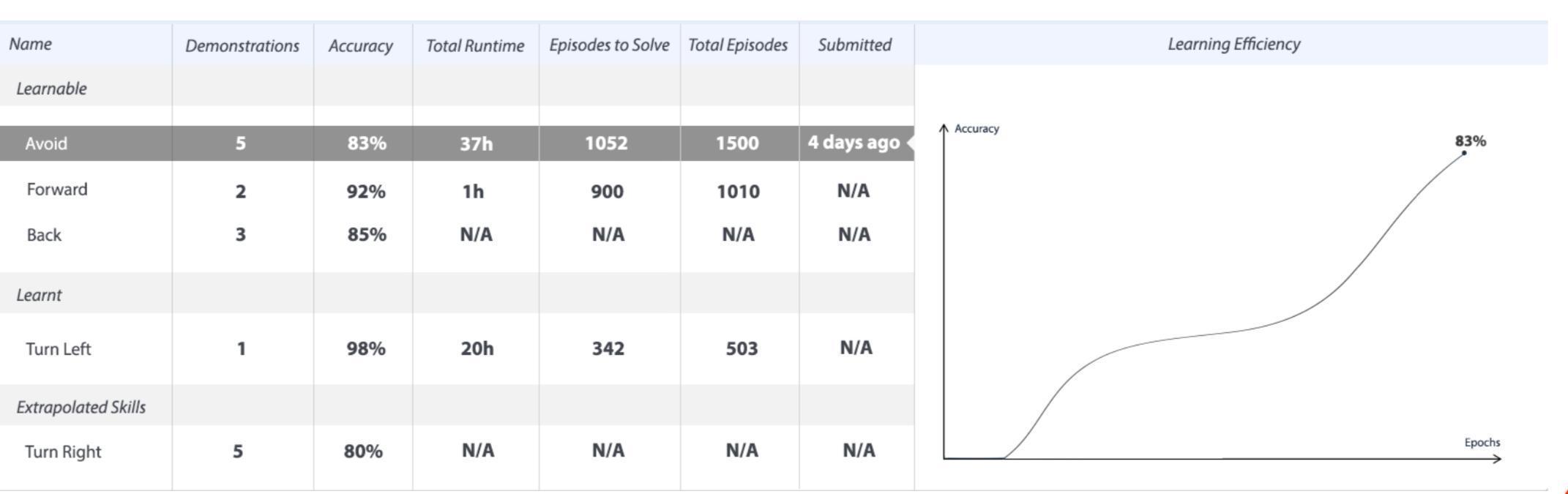


Fig. 2 Visualization Screen

Question:

B) The images in the screenshot are not clear. Sir, can you please send the doc version of the task?

Visualization can be generated at the end of several demos, but what to visualize is still unclear. The actions are grouped for study, means you want to monitor action-wise, am I correct?

Answer:

Those are nothing else but statistics.

Column *Name:* Name of a saved skill by an operator.

Column *Demonstrations:* Number of added demonstrations by a human.

Column *Accuracy:* Accuracy or Confidence of current skill. Not sure how you do it, please propose.

Column *Total Runtime:* Total time spent on training or optimizing of a skill.

~~Column~~ *~~Episodes to solve:~~* We can skip it for now. Instead, let’s add a column *Assets Updated:* the total number of updated/optimized & saved skills.

Column *~~Total Episodes:~~* We can skip it for now. Subcomment: See my comment below.

Column *Submitted:* # of days when a skill was saved or updated.

Please propose any other stats if you feel those might be useful. Not more than 5.

Then in the Column *Name,* we have a few categories of saving skills, i.e. Learnable, Learnt, and Extrapolated Skills. We can skip this part, for now, to make it quicker.

Question:

Column Name:

How do we consider “Forward”, “Back”, “Avoid”, “Turn Left” and “Turn Right” as skills?

We suppose, “Avoid Red Obstacle”, “Avoid Yellow Obstacle”, “Avoid Brown Obstacle” as skills. Subanswer: You are right, it’s just a mockup with examples. In fact it’s up to a person to name a particular “folder” for skills which e.g. you and your colleague will be demonstrating and it depends on the environment, needs, and creativity I would say.

Column Demonstrations:

This we can keep on updating based on the demo’s performed for a particular skill.

Column Accuracy:

We can fix a value for the steps required for the toy car to achieve a goal from point A to B.

And we can calculate the steps actually taken by the toy car during the demo, based on that we can calculate how well it has learned. We will get the accuracy.

Subanswer: Sounds good.

Column Total Runtime:

This we can save

Column Assets Updated:

We can keep a total count of unique assets(skills) demonstrated.

Subanswer: Good idea so maybe we can add Total Episodes Column as well, a few lines of code but actually might be helpful.

Column Submitted:

We can keep track of this too.

Proposed Stats:

1. Latest Feedback Subanswer: Just to make sure, what that would present? If you mean “not ok, can’t tell, ok” than it’s a good idea or maybe not the latest but ratio? i.e. 5% not ok, 85% ok, 10% can’t tell.
2. Latest Epochs (As we will be plotting the graph on Epochs Vs Accuracy) Subanswer: Definitely. Yes.

Re:

Then in the Column Name, we have a few categories of saving skills, i.e. Learnable, Learnt, and Extrapolated Skills. We can skip this part, for now, to make it quicker.

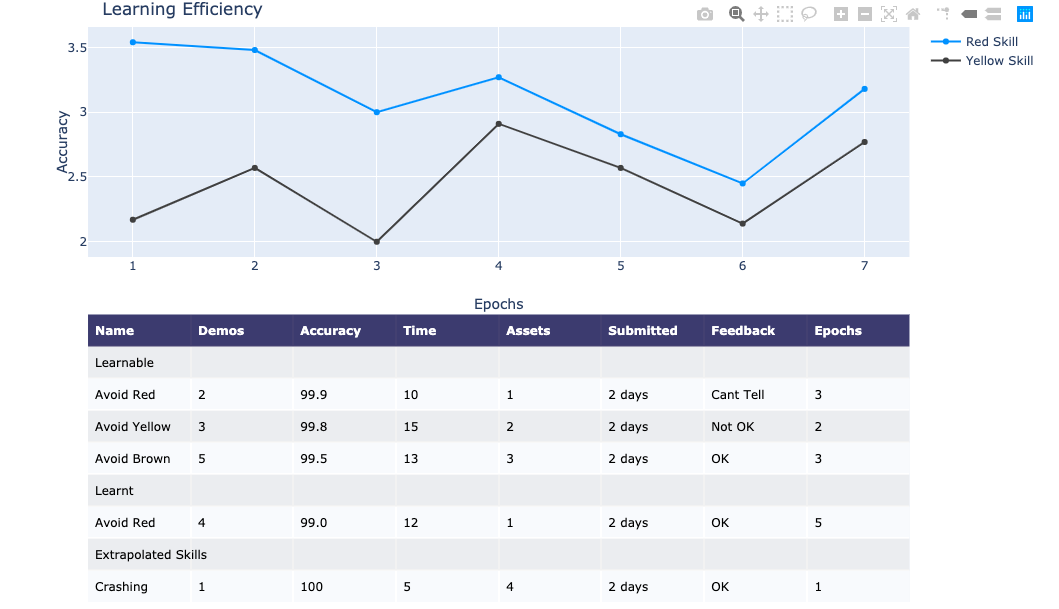
(This is just category name, I don’t thi we need to update anything for these. Still, correct me if I am wrong)

Subanswer: You are right, the idea was to organize some of the skills, so e.g. we know we have a few draft skills, very rough not optimized etc. but we also have a few which are good to go, and we don’t want nor plan to spend time to optimize them anymore. Would be nice if you could suggest a solution here.

Question: Based on our conversation. Do you have any other suggestions or ideas to improve or adjust Task 1? I’m very open for that because people do these kinds of things in many different ways, me myself I saw at least 20 similar approaches for the above task.

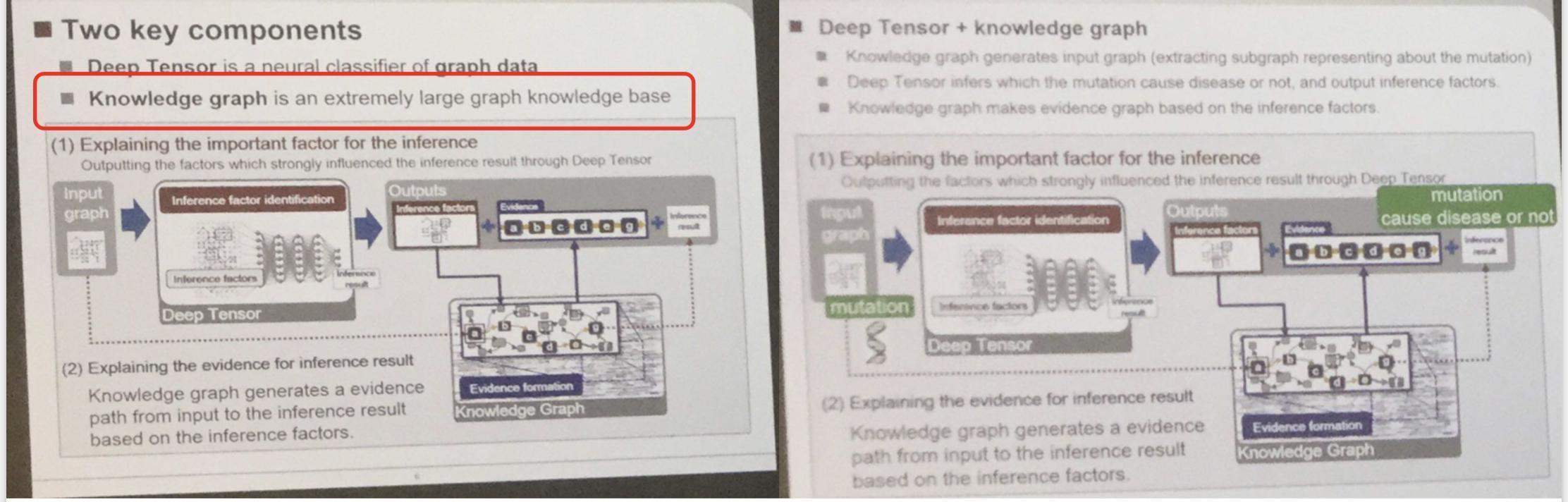
**Work done:**

Please find the below sample skeleton version of GUI, we are working on the dynamic data to be populated in the csv to be used by the code for the plotly GUI.



C) Graph of Tensors

Inspiration, side note:



This might be helpful: <https://neo4j.com/> I would be open if you decide to use this tool.

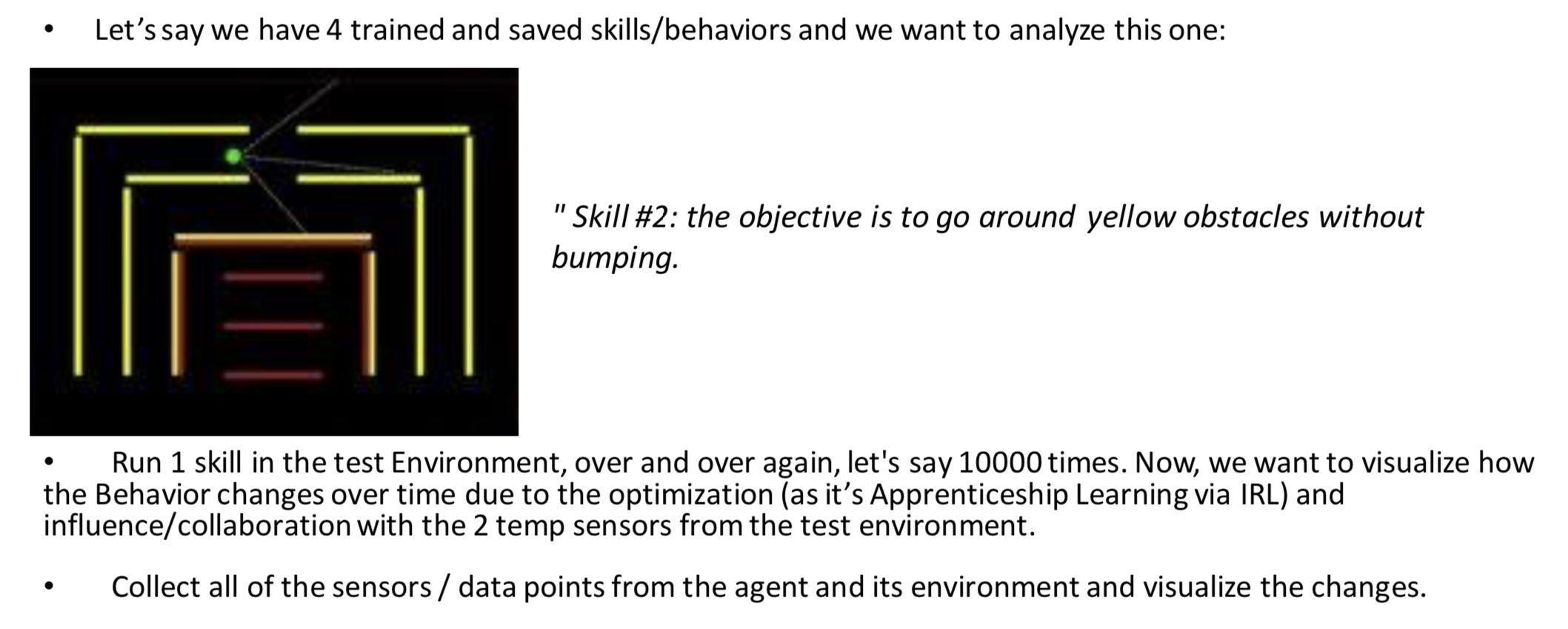
Question:

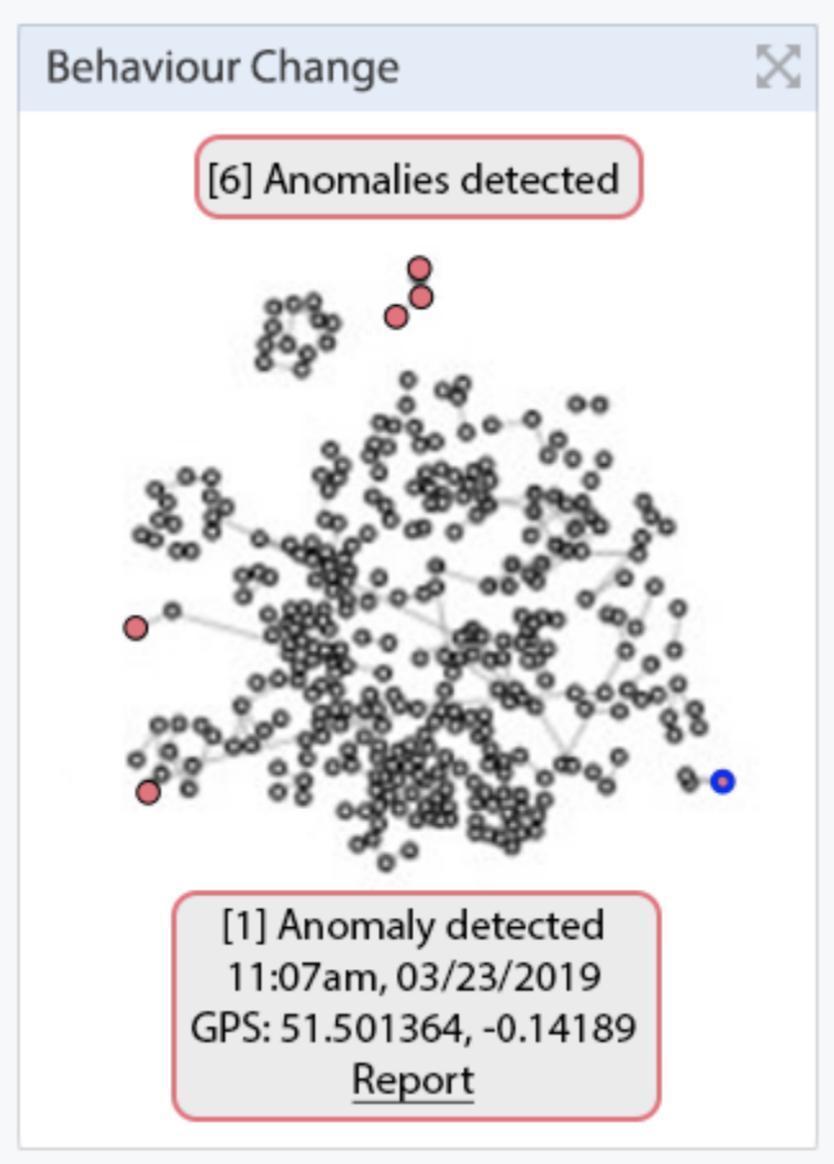
Your aim is to generate a knowledge graph for tensors of what?

Answer: I’m glad you asked that question, thank you; please propose 1 or 2 solutions we can develop or even both if interesting. The idea is broad for purpose, we developed a few approaches already but I think you can get the sense the aim based on our conversation. Hint: see task 2, task 1A.

**Task 2)**

1. Behaviour analysis





Question:

A) You want to carry out behaviour analysis of skill during its optimization in training or playing mode using the trained weights?

Also, exactly what behaviours you want to monitor here?

We can show the collaboration of a particular behaviour of a car, like position or angle with the sensor values. That’s what you want?

Can you define, what is required? A scatterplot to represent the anomalies? For that can you please tell what should be the x and y terms.

Answer:

Re You want to carry out behaviour analysis of skill during its optimization in training or playing mode using the trained weights?

I was thinking about both.

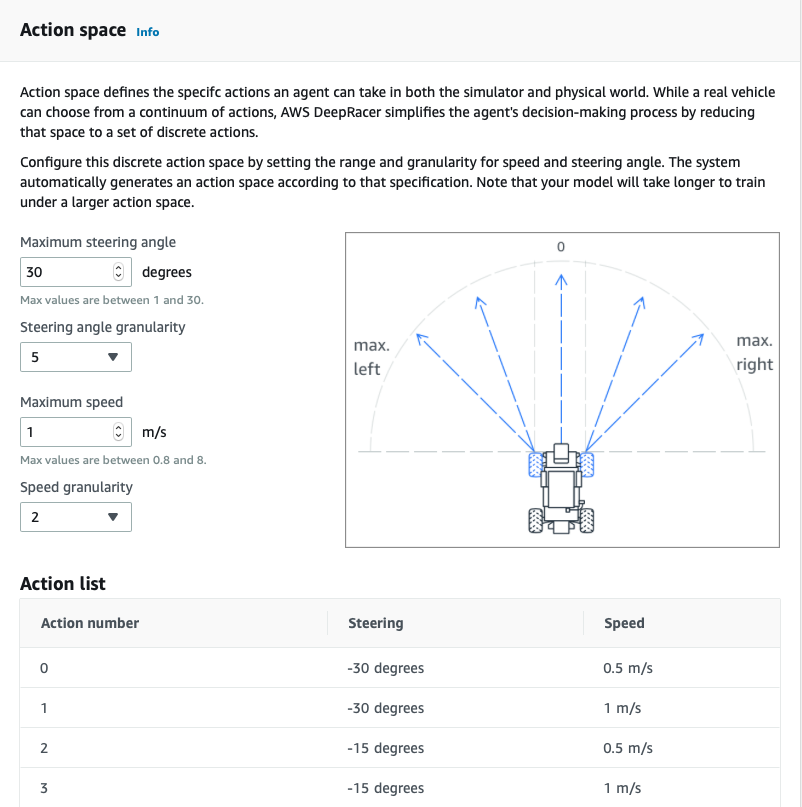
Re Also, exactly what behaviours you want to monitor here?

Usually, people analyse saving skills and the integration of multiple demonstrations of skill into an optimized skill.

Re We can show the collaboration of a particular behaviour of the car, like position or angle with the sensor values. That’s what you want?

I would like to see how a behaviour/demonstration change over time. How different it is after adding multiple demonstrations and most importantly adding optimization (Apprenticeship Learning via IRL). Most probably Bayesian part should be included here but I have to check the previous script.

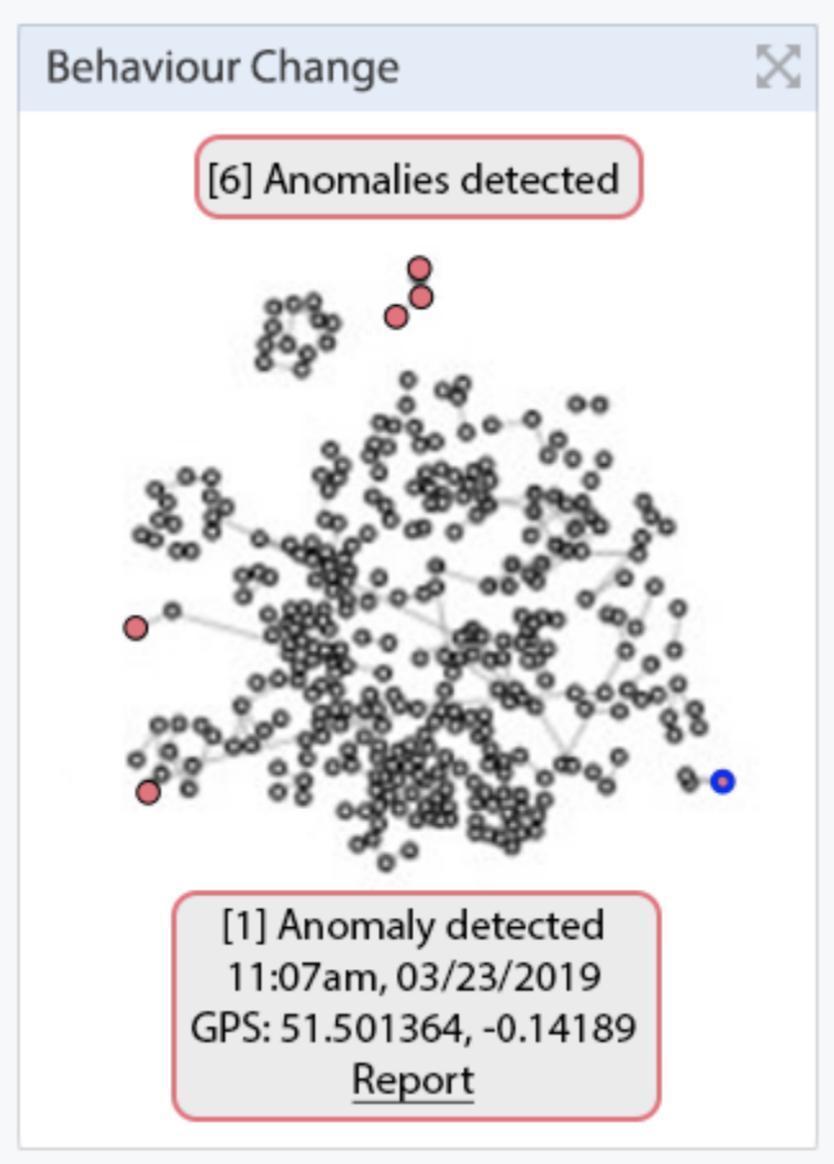
Subanswer: Another dev suggested to add position and angle with the sensor values as well, and he gave this as an inspiration:



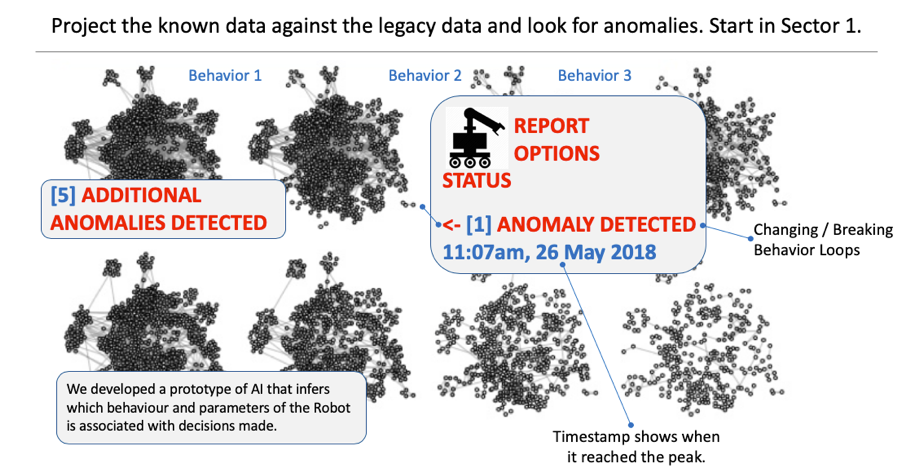
Would be cool to add something like you (and him) proposed; nice concept.

Re Can you define, what is required? A scatterplot to represent the anomalies? For that can you please tell what should be the x and y terms.

Scatter graph might be too simple to represent skills. In the below screenshot someone was presenting behaviour loops classification map. The goal is to find outliers, agent unplanned activity recognition and skills/behaviours’ change pattern discovery.



This might be helpful:



Question:

Scatter graph can be created we can work on that.

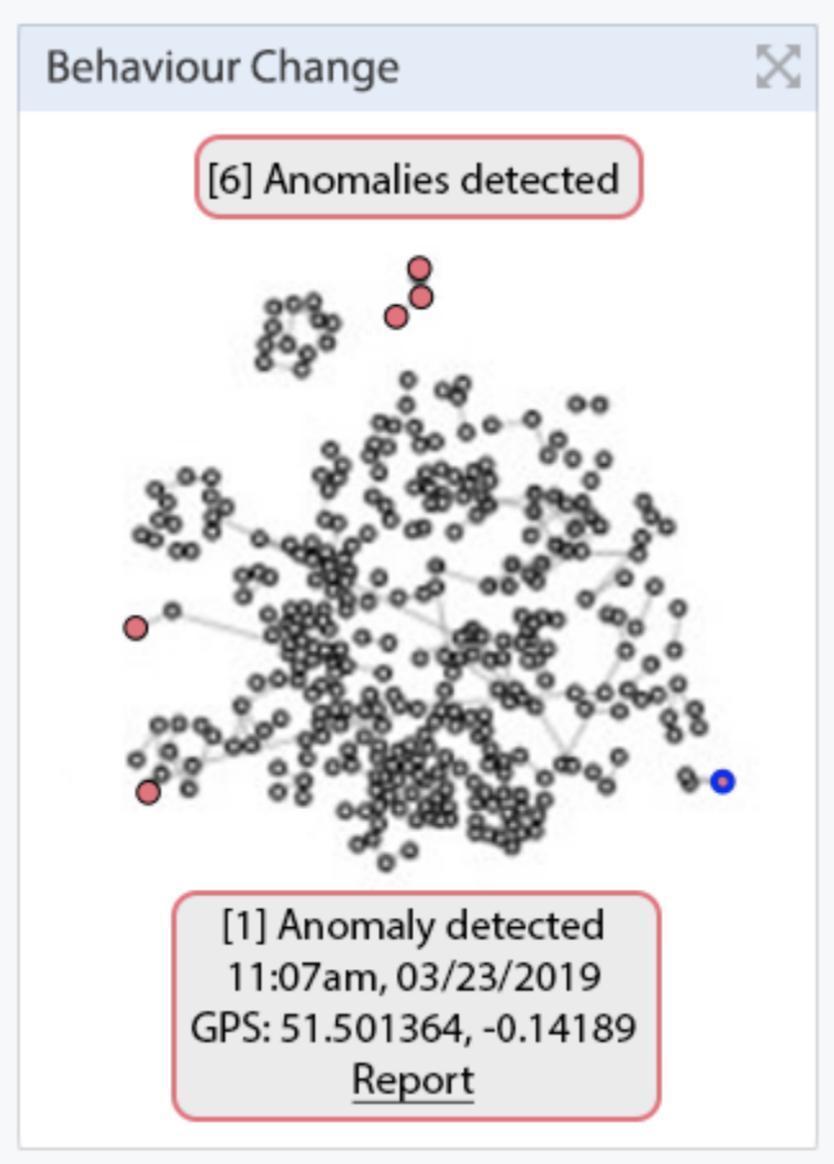
Subanswer: Interesting, didn’t know we could have the same view using such a simple thing like scatter graph. Great.

It will be able to know how the agent has learned. So during training it will show more anomalies initially but will reduce slowly as it learns with time. While in playing the scatter graph will learn how it works after completing the training.

Can you sir please provide a reference of the above screenshots you have shared so we can go through that papers/blogs/links? It would help in smoothen the process. Thanks

Subanswer: Exactly. However, the primary concept was to observe not only the training phase (which is interesting, let’s do it) but also the already learned, trained skills which are deployed in an environment without any optimization nor training. Hence the proposed Column *Name,* with a few categories of saved skills, i.e. Learnt, in Task 1B.

So while an agent is running in an environment I want to see dynamically changing behaviors (for eg. red behaviour). I want to be able to click on a black point/dot (see the map below) to get more information (popup or whatever is possible) such as time, behaviors name and status/values of all involved parameters, i.e. distance sensors, angle etc. In other words, we visualize car/agent behavior in space and time including agent movement analysis, visual, and multimodal analysis of agent behaviour and all kind of environmental conditions.



This mockup was proposed by another dev who wants to do this task. What else can you propose here based on the another dev’s mockup (big one with proposed misbehaviors, comparisons to legacy data, and behavior loops) and your ideas? I’m very much open to implement your propositions as well, please think about it and let me know how might we improve the concept.

**Demo for Task 2) Part A)**

**Behaviour Analysis of BHIRL Implementation**

**Libraries used:-**

1. **Matplotlib :-** for displaying graphs and making analysis for the behaviour of toycar BHIRL while training the model
2. **Pandas :-** Working with csv files in storing and fetching data from the toycar BHIRL while training the model and after that using the csv file to display graphical results.

**Purpose :-**

To view the behaviour analysis in different behaviours (red, brown, yellow), how it is trained and when the toycar crashes and how the learning process takes place with the help of a graphical representation.

**Methodology :-**

For making behavioural analysis of toycar BHIRL following steps are followed and changes were made in the following files:-

i) learning.py ( Codes were added and are commented with Task 2) part a) for reference, after running this code for a particular behaviour a csv file will be created with name output.csv )

ii) output.csv ( csv file with time and car distance at which the toycar crashes in a particular behaviour )

iii) behaviour\_analysis.py ( contains the process of creating the graph for analysis of each behaviour, after running this code result.png file will be created showing graphs for a particular behaviour)

iv) result.png ( Graph showing points at which the toycar crashes for a particular behaviour)

**Steps Followed:-**

**i)** Open learning.py file and train it for particular behaviour, the toycar BHIRL will start training. write the following command in the terminal:-

**python3 learning.py**

**ii)** During learning process the toycar BHIRL will crash many a time at some distances where it fails to travel without crashing.

**iii)** The car distance and time are recorded when it crashes. This data we store it in a csv file and name it output.csv

**iv)** After learning.py completes it training the toycar BHIRL is trained for a particular behaviour and an output.csv file is created.

**v)** After that run behaviour\_analysis.py file in the terminal

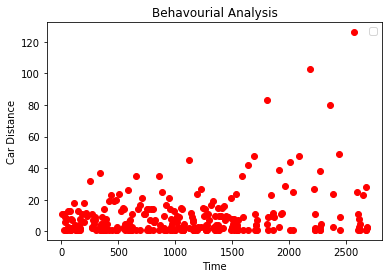
**python3 behaviour\_analysis.py**

**vi)** behaviour\_analysis.py will create a graph based on the data from the csv file and will show the graphical representation when the car crashes while training.

**vii)** result.png file will be created in which the graph will be stored.

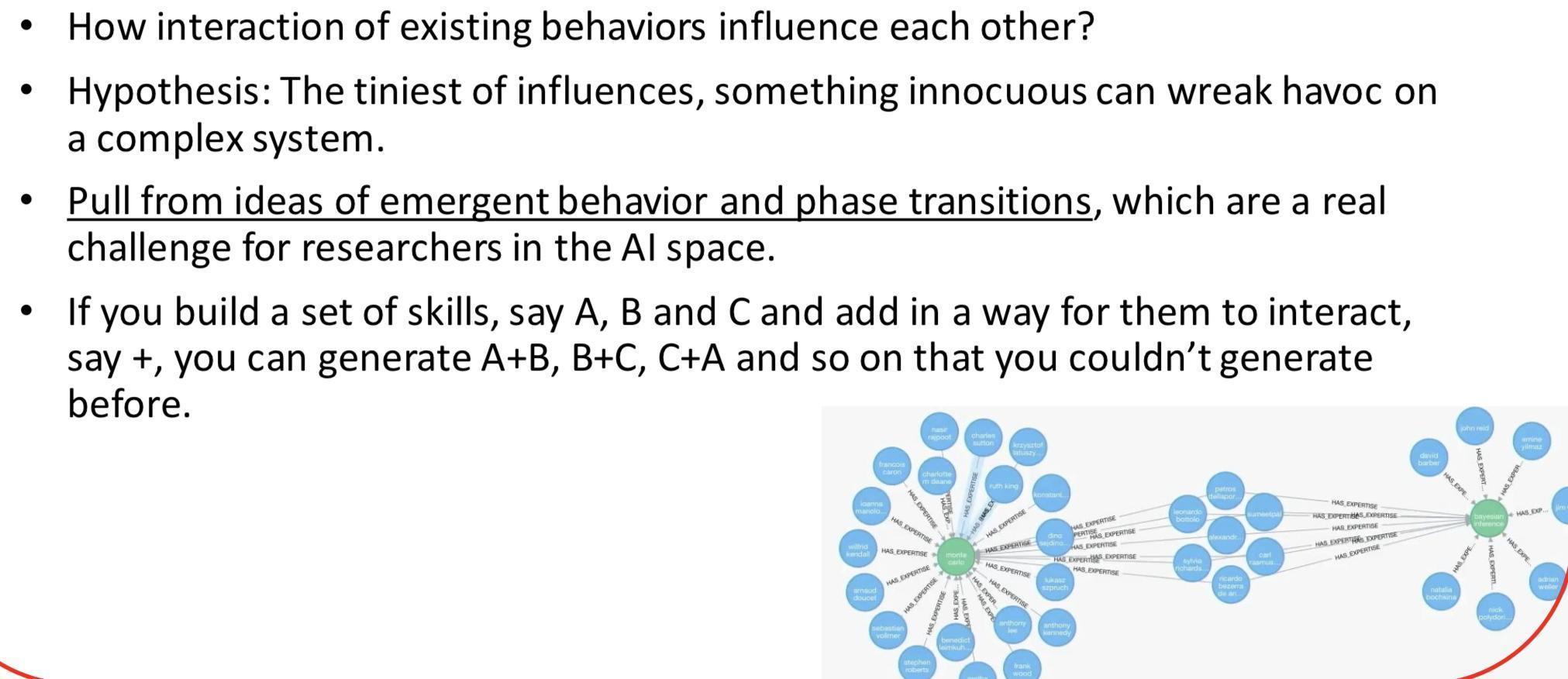
**viii)** The behaviour analysis using the graph will be helpful in understanding how the agent has learned under various parameters and how fast or slow it can learn a particular behaviour.

Snapshot for Behaviour Analysis of Red Behaviour



**PS: please change the names of csv file in learning.py and name of png file in behaviour\_analysis.py for different behaviours each time you train otherwise it will overwrite the files. :)**

B) Behaviour Relationship



Question:

B) This is not clear.

What influence is to be recorded between any two skills? Like when we run a RED behaviour skill and Yellow behaviour skill together, what happens?

I am not sure if this can be done, as two skills can’t be run in parallel for a single toy car.

Answer:

A few ideas: 1) Utilizing skill A+B tend to generate situation X

2) Utilization of skill A+C leads to an elimination of B skill usage.

3) How about we run 2 agents?   
I hope this brings some ideas.

Question:

Sir two skills will not be able to run in parallel.

Subanswer: Of course not, you are right. I meant a sequence of skills, A->B (B after A) etc. Another proposed by a dev concept was to have 2 agents, i.e. Agent1 is executing skill A, while Agent2 is executing skill B. Maybe instead of integrating another agent it would be good to have 1 agent with a bit more developed parameters, e.g. if an agent/car is executing skill A AND while doing so has a specific (proposed by you) ANGLE, then it influences behaviour B.

What would you like to do? I’m flexible especially in this initial few gigs where we warm up, get to know each other, and learn the environment.

We will have to merge two behaviours and work on that (Correct us if we are wrong). Also, please provide reference paper or link for this it will help both

Answer: How do you imagine merging two behaviours and work on that? No one proposed that before, you got me interested. Connection / linking of 2 behaviors is popular and typical (so B, after A etc) but merging two (to have behaviour AB) is something interesting.

**Task\_2\_b WorkFlow:**

Here, we have created three different behaviours by merging yellow, red and brown behaviour as follows:-

1) Yellow\_Red Behaviour

2) Yellow\_Brown Behaviour

3) Red\_Brown Behaviour

Motivation behind merging two behaviours was to understand how the agent performs when it is trained in both the behaviours.

We used three combinations as mentioned above to study and analyze different movements of the toycar when it is working in merged behaviour.

We will look individually steps performed for following learning of all the three combined behaviours:-

**1) Yellow\_Red Behaviour:-**

In the given playground area, we divided it into eight subpolicies for agents to learn and avoid crashing the yellow and red behaviour.

We manually train the agent by making changes in the manualContol.py and obtaining weights for each subploicy.

Following are the steps at which value has been changed in the code and also the values of the 8 subpolicy weights obtained.

Following values needs to be replaced for manually creating Yellow\_Red Behaviour subpolicies in carmunk.py file

Replace the following lines in the code with following values as required:

This code can be found at Line 39 and Line 134 in the carmunk.py file.

Line 39

# Create the car.

self.create\_car(150, 20, 15)

Line 134

self.car\_body.angle = 1.4

change the matrix with any of the following values and you will find the toy car relocated when you run ManualControl.py file.

Then guide it manually and save the weights of each of the eight sub policies.

1) (150, 90, 15) angle=1.4

2) (150, 650, 15) angle=45

3) (950, 550, 15) angle=75

4) (850, 450, 15) angle=400

5) (250, 90, 15) angle=45

6) (250, 450, 15) angle=75

7) (275, 70, 15) angle=650

8) (500, 450, 15) angle=75

Videos of each subpolicies trained manually using manualContol.py file is also attached along the folder.

Subpolicy weights of yellow\_red behaviour

#Yellow\_Red

#weights1 = [11652.97961532, 3485.28413042, 12282.39687071, 2399.60038426, 6707.39188801, 0., 6824.0138868, 0.]

#weights2 = [12031.28573896, 5419.10727811, 7781.1400124, 2132.72097908, 6818.83310555, 0., 6991.19670654, 0.]

#weights3 = [12967.58314534, 7410.86659565, 8499.74756889, 470.00234186, 3307.63194834, 30.93505122, 11603.11138576, 0.]

#weights4 = [12586.132007, 2304.88747211, 10812.66438951, 2419.11783494, 956.13530369, 4913.86688615, 7621.9964967, 0.]

#weights5 = [12743.14355538, 7770.24299784, 9350.64798877, 881.3639501, 14983.99461895, 0., 40.36135716, 0. ]

#weights6 = [1.24235415e+04, 8.50571291e+03, 9.60796656e+03, 4.07788882e+02, 1.53639437e+04, 7.74352438e-01, 1.70243844e+02, 0.00000000e+00]

#weights7 = [ 9502.54353308, 7529.45366433, 7568.10392051, 1469.28794262, 13938.41670243, 0., 58.77659731, 0.]

#weights8 = [13119.90895308, 6400.97060503, 10154.60393634, 913.36095425, 11620.25905423, 99.08305445, 3091.08456729, 0.]

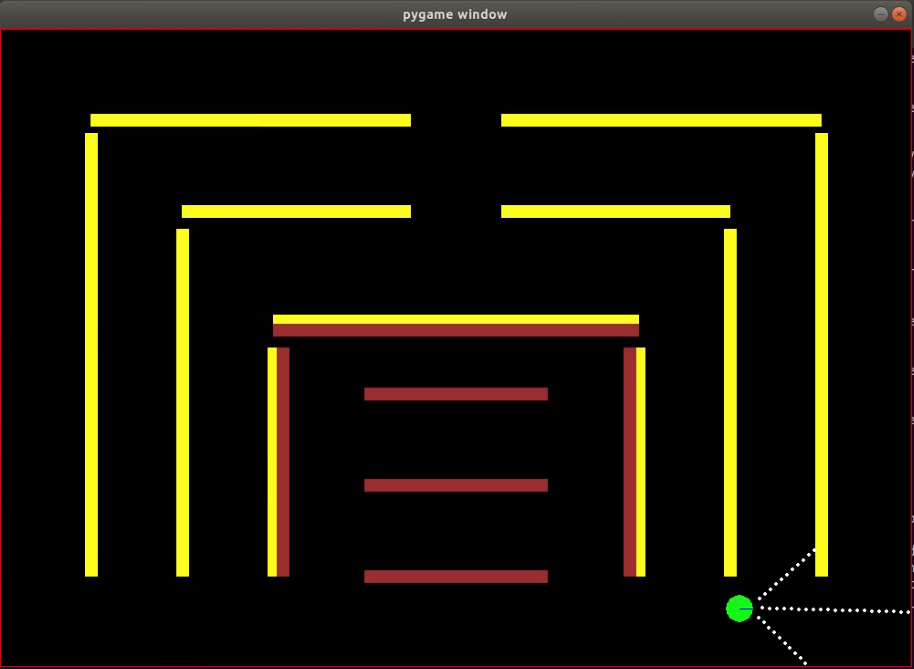
After obtaining the 8 subploicy weights we run normalize\_weights.py file and obtain a normalize weight from the given 8 subweights.

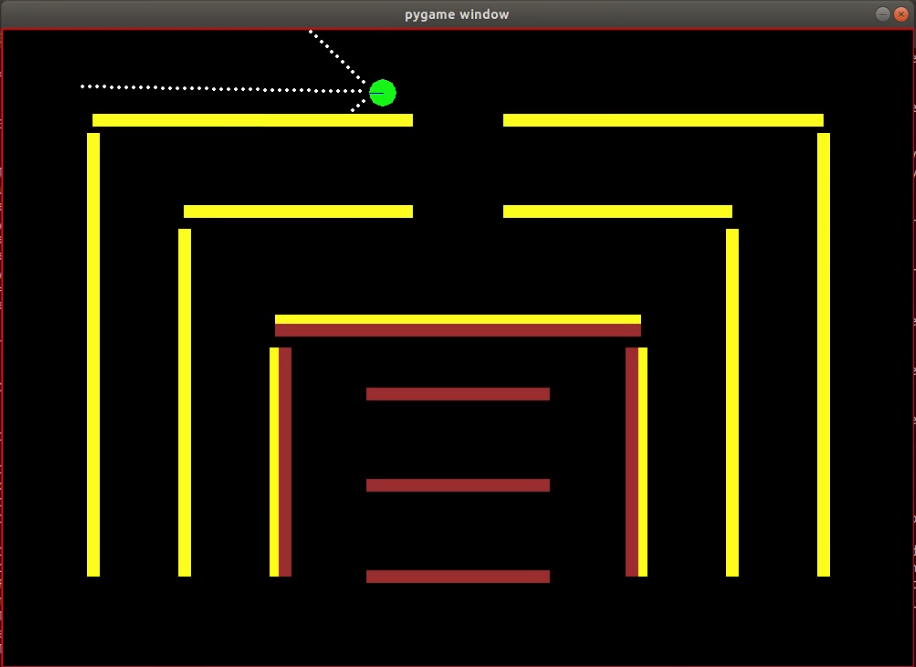
This obtained normalized weights is then copied into learning.py and agent is made to learn using this normalized weight.

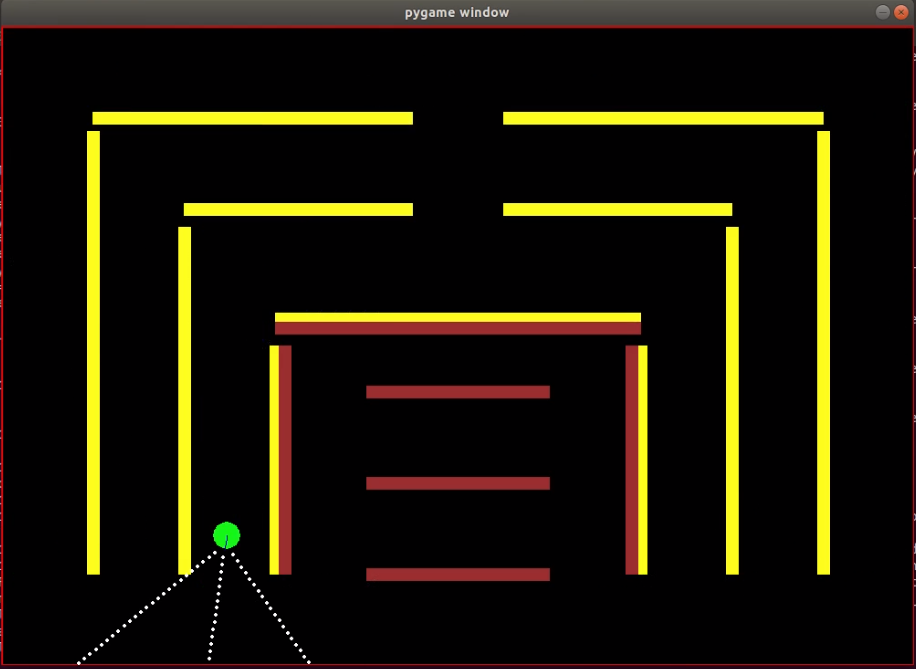
After that the learned weights are saved into saved\_models\_yellow\_red --> EvaluatedPolicies folder in .h5 format.

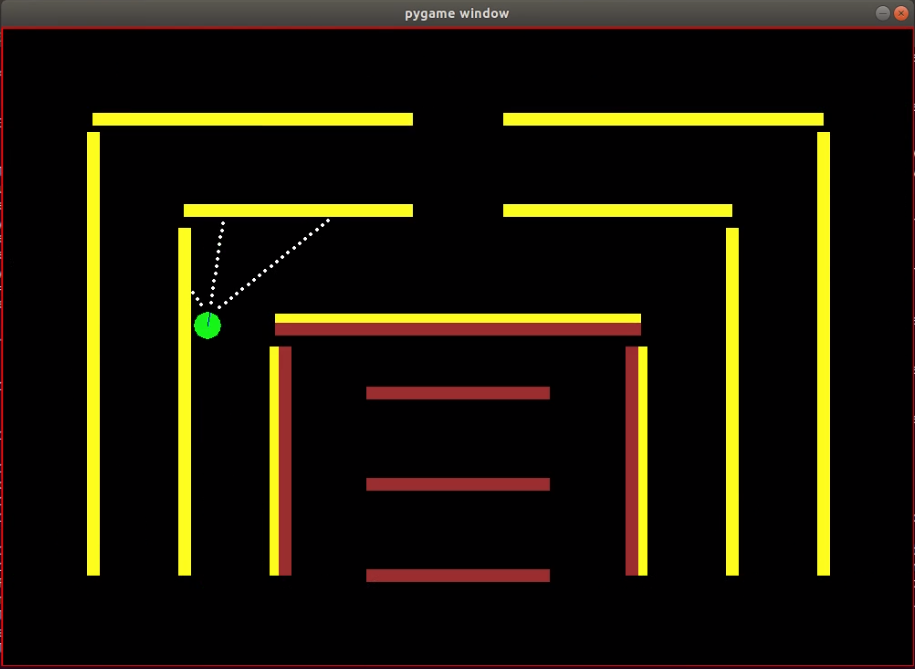
Videos of taking subpolicy weights manually are in saved\_models\_yellow\_red to get an idea of how it is done.

Snapshots of yellow\_red behaviour:-









C) Future Trajectory of a Toy Car/Agent & Obstacles



Question:

Sir, this is incomplete, can you please share the complete doc.

I guess here you want to predict the future track and obstacle (moving?) position.

Answer:

We present future steps/movements of an agent (as many as possible), along with its confidence. It has to be dynamic, so after one step further, it updates the future movements.

Question:

Sir we can work on this using the previous work we did for you. It will help in predicting future steps of the agent. We can work on it.

Subanswer: I know, wanted to see how you will react. :)

Also one general question

Somewhere you have mentioned IRL and some place bayesian, so can you please specify in each task like what approach ( inverse RL, hierarchical RL or Bayesian RL ) to carry forward?

Thanks

Subanswer: I understand we use Bayesian hierarchical Inverse RL as that’s what you developed last time. So we build on top of that for a while to practice. Correct?