Chapter 1

Conclusion

Many aspects of the OceanQA dataset and the H-PERL approach have been discussed in previous chapters. This chapter therefore aims to summarise these details, and particularly focus on the strengths and weaknesses of our VideoQA architecture. We also attempt to compare our approach qualitatively to existing work in VideoQA. Finally, we conclude by outlining a number of possible extensions to our architecture, some of which may help to address the drawbacks of the approach, or apply the approach to different tasks.

1.1 Summary of H-PERL

In Chapter **??** we described the H-PERL architecture as a hybrid architecture - composed of both neural networks and symbolic reasoning modules - made of a pipeline of components. Each component in the architecture extracts a particular concept from the video or question. These concepts include: objects, including type and position; object properties, such as colour and rotation; binary relations between objects, such as close; and events between two consecutive video frames.

Chapters **??** and **??** each outlined specific H-PERL models. The hardcoded model, outlined in Chapter **??**, used mostly manually engineered components. These components allowed the hardcoded model to find and correct errors in the object detection, but also made it less adaptable to new environments. The trained model, described in Chapter **??**, continued to use some engineered components, but trained its *core* components using QA pairs. The trained model also proved the usefulness of hybrid models, by allowing background knowledge of the environment to be used to find the set of effects which occur in a video.

In Chapter **??**, which outlined the performance differences between the models, we saw that the trained model was significantly faster than the hardcoded model, and that the error correcting abilities of the hardcoded model made it slightly more accurate. We also noticed that both models we heavily reliant on highly accurate object detection, but that the trained model was less susceptible to a reduction in accuracy due to noisy data.

Chapter **??** noted that one of H-PERL's key strengths was its adaptability to other tasks; as long as there exists QA pairs with which a component can learn a particular concept, many different QA tasks can be solved. Chapter **??** also mentioned a number of requirements that each H-PERL model has, as well as a number of assumptions that needed to be made about the data. These requirements and assumptions could be deemed quite strict, however, future extensions to our architecture may be able to overcome them.

We now summarise the weaknesses of the current state of the H-PERL approach that have been encountered throughout this report:

1. Both H-PERL models require a highly accurate object detector, which, for many environments, is a very difficult requirement to meet.
2. The H-PERL architecture relies on a hardcoded question parser, which limits each model's ability to adapt to new environments, since, in the general case, parsing questions by-hand is almost impossible.
3. Currently, H-PERL relies on datasets containing specific questions which can be used to train components individually. In a general question-answering setting this will not be the case; questions will require knowledge of many different concepts to answer.
4. H-PERL can currently only solve environments which require knowledge of only: objects, object properties, relations between objects and events. This assumption is too simplistic for many environments.
5. Unlike other machine learning methods, an environment specification must be provided to each H-PERL model. For complex environments, this may be very difficult to generate.

Of course, as well as disadvantages, H-PERL also comes with many advantages. We now list the following strengths of the H-PERL approach:

1. Firstly, H-PERL is highly accurate on the OceanQA dataset; with one of the models answering 99.1% of questions correctly.
2. Since H-PERL stores information extracted from a video symbolically, it is very simple to analyse, understand and explain the model's behaviour.
3. The hardcoded H-PERL model was able to correct errors in an earlier component by reasoning about uncertainty. And, indeed, there are many more possibilities for further symbolic error correction functionality to be added, since H-PERL represents information about each video explicitly.
4. Both H-PERL models proved the utility of background knowledge injection. The hardcoded model was able to use an entire $\mathcal{AL}$ model to detect events, and the trained model was able to use ASP rules to detect effects.

5. After symbolic information has been extracted from the video, all of the questions for that video can be answered in one go, meaning the architecture can be quite fast. This is in contrast to some VideoQA implementations which make use of attention; each question is applied to the attention module individually in order to extract the relevant parts of the video.

There may be many more strengths and weaknesses of the H-PERL approach, such as training and evaluation speed or efficiency. However, given that time and resources for the project were limited, it was not always possible to conduct such performance evaluations or quantitative comparisons to other models. We also note that Section 1.3 mentions a number of possible improvements to H-PERL which may help to mitigate some of its weaknesses.

1.2 Comparison with Existing Models

We spend the second part of this section comparing H-PERL to the Neuro-Symbolic Dynamic Reasoning [11] model, since, to the best of our knowledge, it is currently the only other hybrid VideoQA implementation. Firstly, though, we compare H-PERL to some of the existing neural end-to-end VideoQA approaches.

1.2.1 End-to-End Architectures

Both H-PERL and all of the end-to-end neural network approaches presented in Chapter **??** attempt to extract features from the video. One of the key differences between the two approaches is that H-PERL makes these features explicit, while end-to-end networks do not, instead the information in the video is stored implicitly, in a vector. Clearly, storing information explicitly makes the model more transparent, but, in many cases, it is very difficult, especially for the real-world datasets used by many of the implementations in Chapter **??**.

A further difference between H-PERL and end-to-end networks is that H-PERL uses a hardcoded question parser, whereas many existing VideoQA implementations use a neural component, and again store the information extracted from the question implicitly. H-PERL's use of a hardcoded question parser makes it less adaptable to other datasets. It also makes applying H-PERL to free-form, natural language questions without templates impossible.

Finally, attention mechanisms have been used by a number of recent VideoQA implementations [5, 9, 10, 7, 2, 12, 1, 6]. These allow the model to focus on the most relevant part of the video, based on the current question. While object detection may be considered an attention mechanism in H-PERL, the attention is not dynamically updated for each question. Adding an attention mechanism to H-PERL could allow the model to focus on only the relevant objects of a video, however, a lack of attention allows H-PERL to answer every question for a given video at the same time. Adding an attention mechanism may therefore slow each model down. We consider this idea further in Section 1.3.

1.2.2 NS-DR Model

The Neuro-Symbolic Dynamic Reasoning (NS-DR) model was outlined in Chapter **??**. We now compare this model to the H-PERL models described in previous chapters.

At a high level, the Neuro-Symbolic Dynamic Reasoning (NS-DR) model, which was outlined in Chapter **??**, is very similar to H-PERL. Both approaches make use of some form of object recognition (NS-DR uses Mask R-CNN [4], while H-PERL uses Faster R-CNN [8]), both models extract attributes or properties of objects using neural networks, both approaches attempt to extract event information from the video, and finally, in order to find answers to questions, both models execute programs with the data extracted from the video.

However, when the approaches are analysed in depth a number of key differences emerge, which we list as follows:

- NS-DR uses a single 'object-centric' component, the video frame parser. This component roughly corresponds to two different H-PERL components: the object detector, properties component. The NS-DR video frame parser is also pre-trained on non-QA data - data other than QA pairs. Whereas, in H-PERL, only the object detector is pre-trained; the properties component can be trained using QA pairs. In this sense H-PERL is more adaptable to new environments, since each environment requires only a trained object detector and a dataset containing properties questions.

- While H-PERL makes use of symbolic event components, NS-DR uses a neural event prediction engine which models objects and relations between objects as a graph. This likely makes NS-DR's event detection less transparent, since it is harder to tell how the model came to a particular conclusion, but may improve accuracy.

- NS-DR uses a neural question parser which constructs a functional program, with which the output of the event prediction component is executed. Training this question parser requires additional supervised data, in the form of pairs of natural language questions and their corresponding functional program. H-PERL's question parser is manually engineered specifically for the OceanQA environment. In this case, both question parsing components would be difficult to adapt to new environments, since one would need to either engineer a new component or have a dataset of questions and functional programs available.

- Finally, we note that the methods for answer generation are similar in the two models, except that NS-DR uses a trained question parsing component to generate a program to be executed, whereas H-PERL uses a manually engineered question-anwering component, and that H-PERL uses logic programming to both represent the video information and to find the answer to the question.

1.3 Future Work

The two sections above indicate a number of possible extensions to the H-PERL model, which may help to alleviate the approach's the drawbacks. We outline some of these extensions as follows:

- Firstly, one of the major drawbacks of the model is its inability to work with non-templated questions and its use of an engineered question parser. Future models may be able to extend H-PERL by adding natural language processing (NLP) components which can take any question and either extract some symbolic information, or even produce an ASP rule which can find an answer to the question, in a similar way to the NS-DR model.
- In addition to an NLP question parser, a neural attention module could be added, which would allow the model to focus on the most salient aspects of a video, for a given question.
- In order to learn rules for answering questions, Inductive Logic Programming (ILP) could be applied to the QA system component. This would enable the QA component to be easily applied to new environments and dataset.
- Another helpful extension to the model would be to allow end-to-end training to take place, rather than relying on training each component individually using curriculum learning. An end-to-end trained model, may include a *causality* module which, for a given question, would allow it understand which component of the model was responsible for an incorrect answer and should be updated.
- A number of extensions to the OceanQA dataset could increase the diffuculty of the environment. These could include: allowing mutliple moving objects; additional object properties such as velocity or acceleration; including non-deterministic actions; adding more binary (or higher arity) relations; or allowing events which need to modelled using continuous variables, moving by x pixels, for example.
- Finally, questions which require a deeper understanding of the environment could be added to the dataset. These could include the explanatory, predictive or counterfactual questions used in the CLEVRER dataset, outlined in Chapter **??**. Or questions which ask the model to explain why an event occurred could be added.

Finally, we conclude by saying that hybrid machine learning models are an exciting and growing area of research. And that we hope this work is a helpful contribution to the area, and that it will inspire futher related work on hybrid models.

Bibliography

[1] Chenyou Fan et al. “Heterogeneous Memory Enhanced Multimodal Attention Model for Video Question Answering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1999–2007.

[2] Jiyang Gao et al. “Motion-appearance co-memory networks for video question answering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6576–6585.

[3] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2014.

[4] Kaiming He et al. “Mask R-CNN”. In: *The IEEE International Conference on Computer Vision (ICCV)*. 2017.

[5] Yunseok Jang et al. “Tgif-qa: Toward spatio-temporal reasoning in visual question answering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2758–2766.

[6] Kyung-Min Kim et al. “DeepStory: Video Story QA by Deep Embedded Memory Networks”. In: IJCAI17. AAAI Press, 2017, 20162022.

[7] Jie Lei et al. “Tvqa: Localized, compositional video question answering”. In: *arXiv preprint arXiv:1809.01696* (2018).

[8] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[9] Dejing Xu et al. “Video Question Answering via Gradually Refined Attention over Appearance and Motion”. In: *MM ’17*. 2017.

[10] Yunan Ye et al. “Video question answering via attribute-augmented attention network learning”. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 2017, pp. 829–832.

[11] Kexin Yi et al. “Clevrer: Collision events for video representation and reasoning”. In: *arXiv preprint arXiv:1910.01442* (2019).

[12] Youngjae Yu et al. “End-to-End Concept Word Detection for Video Captioning, Retrieval, and Question Answering”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 3261–3269.

Appendix A

ASP encoding of $\mathcal{AL}$

In this appendix we present a method for encoding an $\mathcal{AL}$ system description in ASP, as described in [3]. We need to encode three different parts of the system description: the signature, the $\mathcal{AL}$ statements and some domain-independent axioms.

We encode the signature of the system description, $sig(SD)$, as follows:

- For each constant symbol c of sort $sort\_name$ other than $fluent$, $static$ or $action$, $sig(SD)$ contains

$$sort\_name(c). \tag{A.1}$$

- For every static g of SD, $sig(SD)$ contains

$$static(g). \tag{A.2}$$

- For every intertial fluent f of SD, $sig(SD)$ contains

$$fluent(inertial, f). \tag{A.3}$$

- For every defined fluent f of SD, $sig(SD)$ contains

$$fluent(defined, f). \tag{A.4}$$

- For every action a of SD, $sig(SD)$ contains

$$action(a). \tag{A.5}$$

In the following we refer to the ASP encoding of the $\mathcal{AL}$ system description as $\Pi(SD)$, where $\Pi(SD)$ includes $sig(SD)$. We introduce a relation $holds(f, i)$ which says that fluent f is true at timepoint i. We also introduce the notation $h(l, i)$ where l is a domain literal and i is a step, which will not be used in the ASP program, but will instead be replaced by either $holds(f, i)$ if $l = f$, or by $\neg holds(f, i)$ if $l = \neg f$.

We encode the $\mathcal{AL}$ statements as follows:

- If the maximum number of steps is $< max >$, then $\Pi(SD)$ includes

$$\#const\ n = \ < max > . \tag{A.6}$$
$$step(0..n). \tag{A.7}$$

- For every causal law, a **causes** l **if** $p_0, ..., p_m$ $\Pi(SD)$ contains

$$h(l, I+1) \text{ :- } h(p_0, I), ..., h(p_m, I), occurs\_action(a, I), I < n. \tag{A.8}$$

- For every state constraint, l **if** $p_0, ..., p_m$ $\Pi(SD)$ contains

$$h(l, I) \text{ :- } h(p_0, I), ..., h(p_m, I). \tag{A.9}$$

- For every executability condition, **impossible** $a_0, ..., a_k$ **if** $p_0, ..., p_m$ $\Pi(SD)$ contains

$$\neg occurs\_action(a_0, I); ...; \neg occurs\_action(a_k, I) \text{ :- } h(p_0, I), ..., h(p_m, I). \tag{A.10}$$

The ; in the head of rule A.10 stands for logical disjunction and can be read as meaning at least one of $a_0, ..., a_k$ must not occur at timepoint I.

In order to complete our encoding of an $\mathcal{AL}$ system description we need to add a number of domain-independent axioms. These axioms are not specific to any system or task, but rather convey commonsense knowledge that should apply to many systems. It is worth noting, however, that in certain situations some or all of these axioms may not make sense and should not be used.

We encode the domain-independent knowledge as follows:

- The inertia axiom states that inertial fluents will keep their state unless explicitly changed:

$$\begin{aligned} holds(F, I+1) \text{ :- } & fluent(inertial, F), \\ & holds(F, I), \\ & not\ \neg holds(F, I+1), \\ & I < n. \end{aligned} \tag{A.11}$$

$$\begin{aligned} \neg holds(F, I+1) \text{ :- } & fluent(inertial, F), \\ & \neg holds(F, I), \\ & not\ holds(F, I+1), \\ & I < n. \end{aligned} \tag{A.12}$$

- The closed-world assumption (CWA) for defined fluents states that defined fluents which are not known to be true are assumed to be false.

$$\neg holds(F, I) \text{ :- } fluent(defined, F), step(I), not\ holds(F, I). \tag{A.13}$$

- The CWA for actions states that actions that are not known to occur are assumed to not occur.

$$\neg occurs\_action(A, I) \text{ :- } action(A), step(I), not\ occurs\_action(A, I). \quad \text{(A.14)}$$

Finally, in order to make use of the system description encoding we would include information on which events occured, using the $occurs\_action(A, I)$ predicate, and information on the different states of the system using $holds(F, I)$ and $\neg holds(F, I)$.

Briefcase Example

We now consider a simple example domain: a briefcase with two clasps (from [3]). There is a single action, toggle, which moves a given clasp into the up position if it is down, and vice-versa. If both clasp are up the briefcase is open, otherwise, it is closed.

The signature of the domain consists of sort clasp, which can take value 1 or 2, inertial fluent $up(C)$, defined fluent $open$ and action $toggle(C)$, where C is one of the clasps.

A schema for the system description is as follows:

$$toggle(C) \textbf{ causes } up(C) \textbf{ if } \neg up(C) \quad \text{(A.15)}$$

$$toggle(C) \textbf{ causes } \neg up(C) \textbf{ if } up(C) \quad \text{(A.16)}$$

$$open \textbf{ if } up(1), up(2) \quad \text{(A.17)}$$

This is known as a schema for a system description since it contains variables. Individual rules can be obtained by grounding the variables. So each of the first two schema rules would produce two ground rules, one where $C = 1$ and another where $C = 2$.

Following the procedure outlined above, we would encode this system description into the following ASP program:

```
% Possible timesteps
#const n = 1.
step(0..n).

% Signature
clasp(1).
clasp(2).

fluent(inertial, up(C)) :- clasp(C).
fluent(defined, open).
action(toggle(C)) :- clasp(C).
```

```
% Domain dependent rules
holds(up(C), I+1) :- occurs_action(toggle(C), I),
                     -holds(up(C), I),
                     I<n.

-holds(up(C), I+1) :- occurs_action(toggle(C), I),
                      holds(up(C), I),
                      I<n.

holds(open, I) :- holds(up(C), I), holds(up(2), I).

% Domain independent rules
holds(F, I+1) :- fluent(inertial, F),
                 holds(F, I),
                 not -holds(F, I+1),
                 I<n.

-holds(F, I+1) :- fluent(inertial, F),
                  -holds(F, I),
                  not holds(F, I+1),
                  I<n.

-holds(F, I) :- fluent(defined, F), step(I),
                not holds(F, I).

-occurs_action(A, I) :- action(A), step(I),
                        not occurs_action(A, I).
```

In a similar fashion to system description schemas, ASP programs allow variables. ASP solvers will first ground the program by computing the the *relevant grounding* - the set of ground rules which could be used by the program. The relevant grounding can be infinite so care needs to taken to ensure that it can be computed.

Appendix B

OceanQA Examples

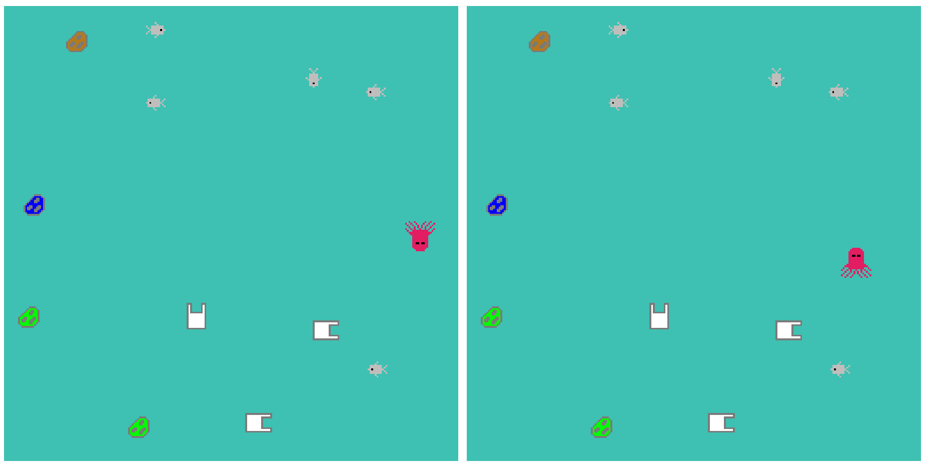
This supplementary material provides a number of example questions and video frames from the OceanQA dataset. The appendix is split into sections based on the question type. Each section outlines a number of example questions and some of the video frames which relate to each question.

Property Questions

Q: What rotation was the brown rock in frame 21?
A: upward-facing

Q: What colour was the right-facing fish in frame 27?
A: silver

Figure B.1 shows frames 21 and 27 from these videos.



(a) Frame 21. **(b)** Frame 27.

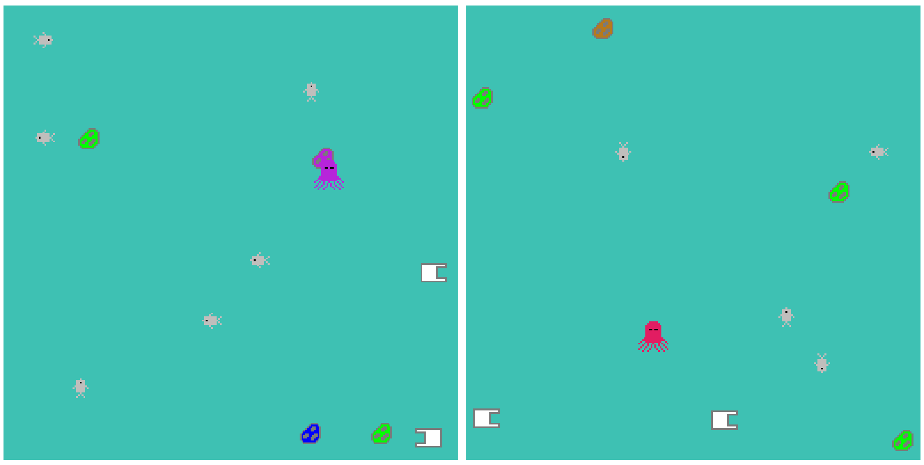
Figure B.1

Relation Questions

Q: Was the purple rock close to the octopus in frame 7?
A: yes

Q: Was the upward-facing fish close to the brown rock in frame 20?
A: no

Figure B.2 shows the two frames these questions are referring to.



(a) Frame 7. **(b)** Frame 20.

Figure B.2

Action Questions

Q: Which action occurred immediately after frame 16?
A: rotate anti-clockwise

Figure B.3 gives an example of an octopus rotating anti-clockwise.

Q: Which action occurred immediately after frame 12?
A: move

Changed-Property Questions

Q: What happened to the octopus immediately after frame 0?
A: Its rotation changed from upward-facing to left-facing

Figure B.3 shows the first and second frame from this video.

Q: What happened to the octopus immediately after frame 20?
A: Its colour changed from red to blue

Figure B.4 shows frame 20 and frame 21 from this video.

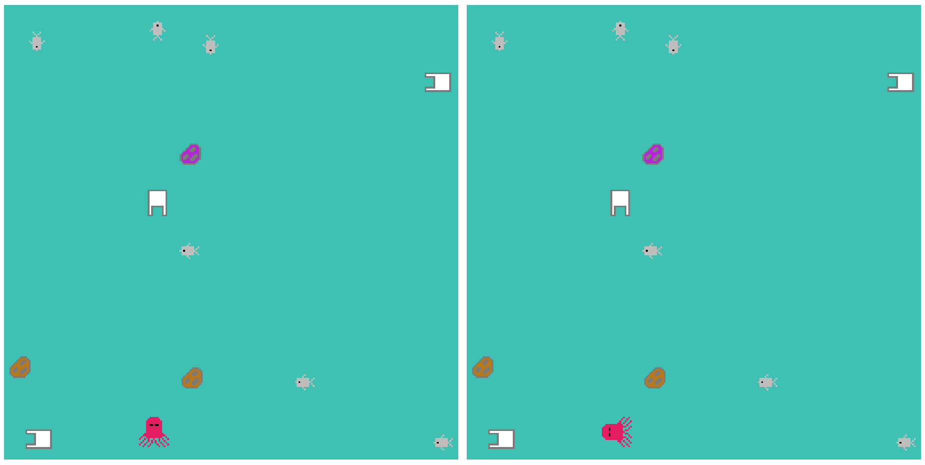


Figure B.3: An example of an octopus rotating clockwise.

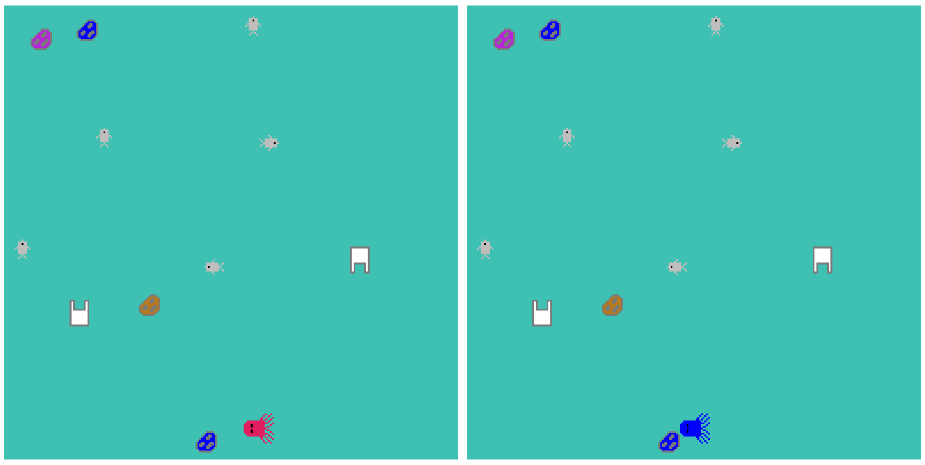


Figure B.4: An example of an octopus changing colour.

Repetition Count Questions

Q: How many times does the octopus rotate clockwise?
A: 0

Figure B.7 gives an example of an octopus rotating clockwise and then moving.

Q: How many times does the octopus eat a bag?
A: 1

Figure B.5 shows the octopus eating a bag.

Repeating Action Questions

Q: What does the octopus do 1 times?
A: eat a fish

Figure B.6 shows an octopus eating a fish.

Q: What does the octopus do 4 times?
A: rotate clockwise

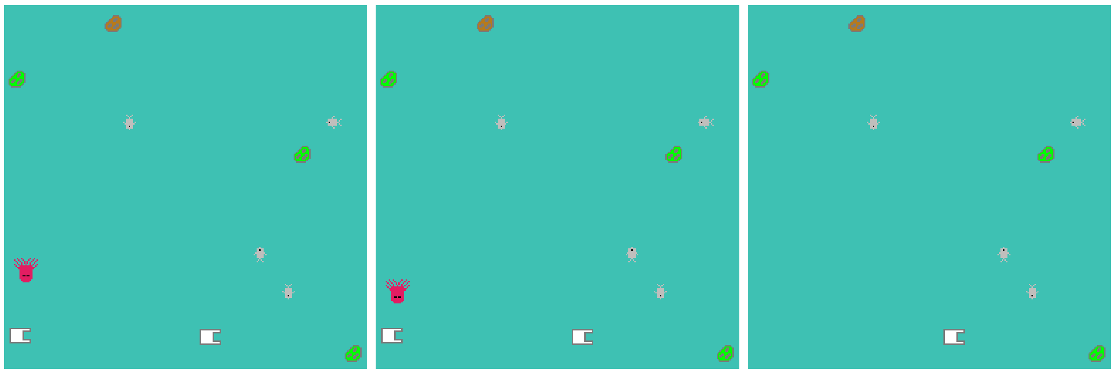


Figure B.5: An example of an octopus eating a bag.

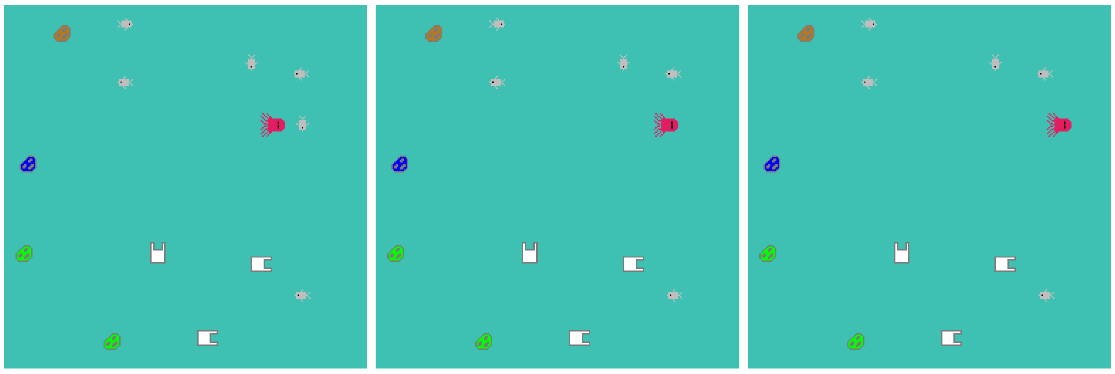


Figure B.6: An example of the octopus eating a fish.

State Transition Questions

Q: What does the octopus do immediately after rotating clockwise for the fourth time?
A: move

Figure B.7 shows the video frames which this question is asking about.

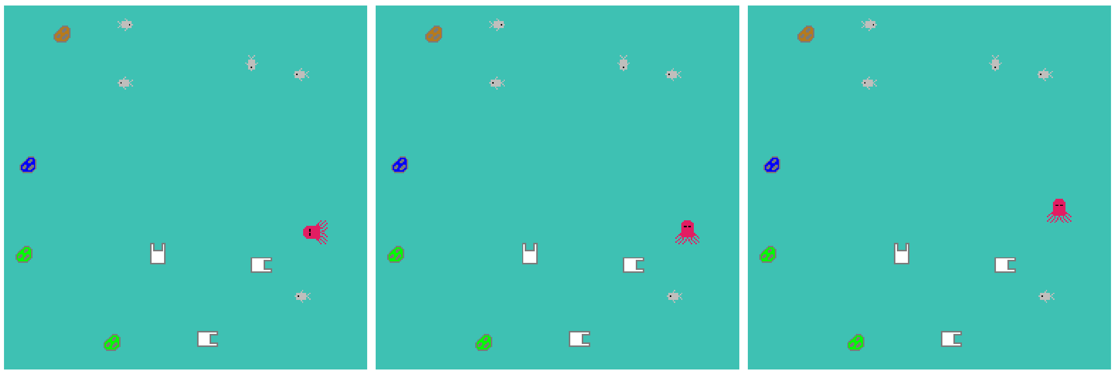


Figure B.7: An example of an octopus rotating clockwise before moving.