Co-operative AI Using ML-Agents

Zala Habib

1 Introduction

In this paper I will discuss the process of studying and designing a co-operative experience for AI within Unity's ML-Agents toolkit[1], as well as the experienced issues and failures in attempting to make this happen. In observing the supplied Sample Environments in the toolkit I noted the absence of a co-operative Agent, which I aim to design and train within a new simple environment. I begun with a base made following a tutorial on how to create a new environment for ML-Agents[2]. This paper assumes an understanding of that tutorial and basic understanding of ML-Agents and Unity.

2 Designing for the Agents

Designing for AI varys quite largely from designing for humans. Humans have a much wider range of perception which can either help or hinder the experience. These perceptions include:

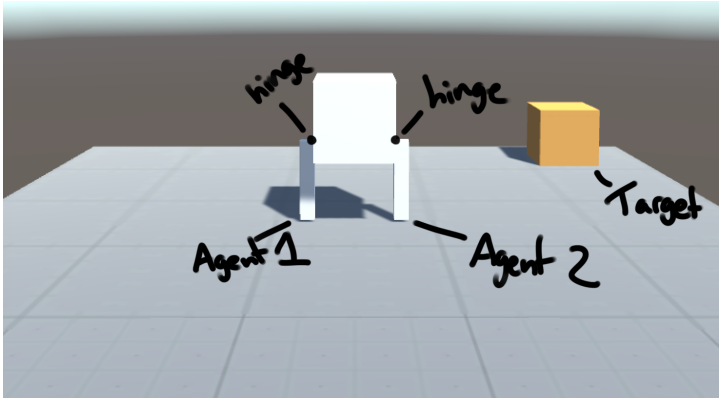
- Aesthetics/graphics - the look of the game
- Reading text/understanding context - a deeper understanding of the scenario
- Emotion/competitiveness - feelings towards the game or other players
- User Experience/User Interface - how the interaction with the game feels
- Understanding control schemes - inherent or confusing control schemes

Some of these make it easier to design for humans but likewise, can make it more difficult or much bigger task. A number of these, such and understanding control schemes, aren't an issue for an AI. This gives it an advantage over the human brain, and makes it easier to design for. Likewise, something such as the context of the environment can be understood by humans and not AI, which may give the human player the upper hand in activities such as problem solving.

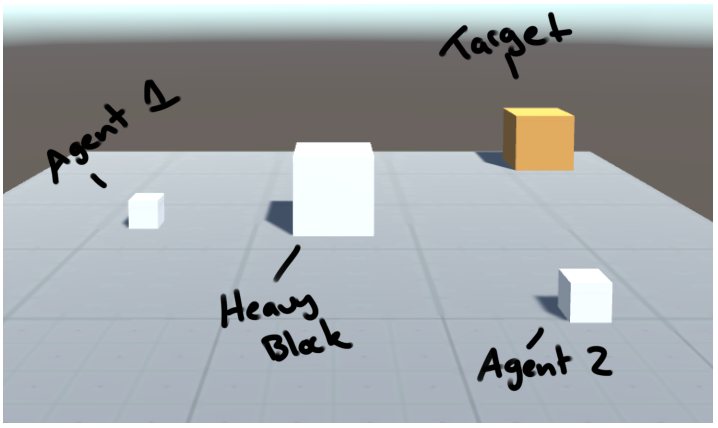
2.1 Concepts

I set out to create a simple environment that utilized co-operative AI in an understandable and achievable setting. The following are a few concepts:

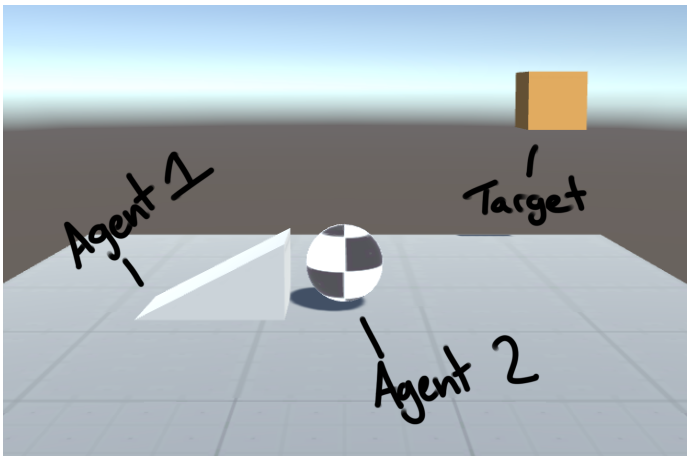
- Walking on hinges
    - This environment consists of a block with two legs attached by hinges. One AI controls each leg and can do nothing except rotate it's leg around the hinge. This environment is feasible with only one AI, though it would need to be trained alongside itself to learn to co-operate with an identical Agent.



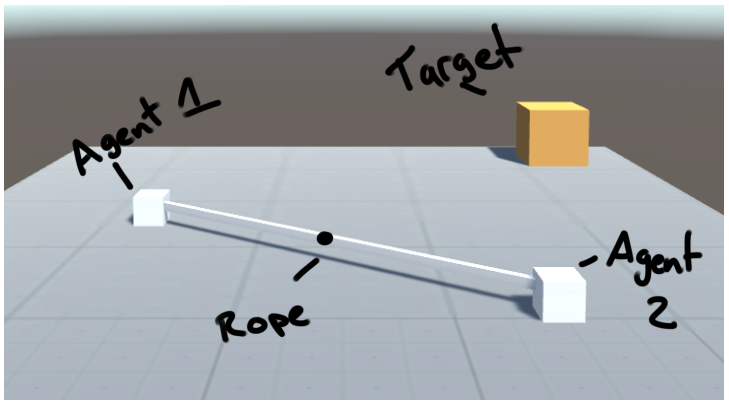
- Pushing a heavy block to a goal
    - This would consist of two identical Agents controlling two seperate blocks with basic movement. The goal would be to push a heavy block to a goal area - except no one Agent alone could push the block. It requires the combined force of both Agents to move the block. This wouldn't require the AI to be aware of it's colleague at all, it would simply require them both to attempt to push the block. This is less interesting due to the lack of awareness between the two Agents - though arguably more doable.



- Ramp and Ball
    - Two AI control two objects - one controls a ramp and another controls a ball. The goal is for the ball to reach a floating target - one it can only reach when it is on top of the ramp. The two Agents need to communicate to reach a common goal, with wildly different actions. This would be interesting to observe as they must each figure out how to work with the other to get the ball to the goal.



- Two AI tied together
    - This involves two identical AI controlling boxes that are connected by a string. Attached to the center of the string is a smaller box - the goal is for this smaller box to reach a target point. This requires the two Agents to work together to get the box to the target.



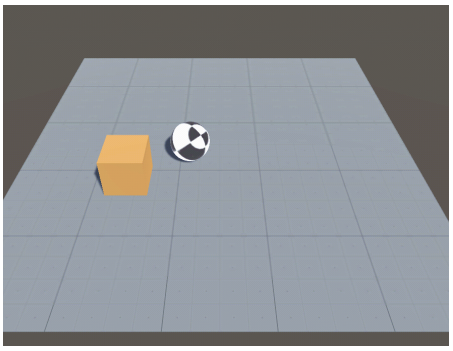
I decided to pursue the Ramp and Ball environment, as this requires two significantly different AI to co-operate and problem solve together. Neither can reach its goal without the other. As a result, it is possibly the most difficult of the concepts to achieve, but also the most interesting to observe, whether or not it succeeds.

3 Controlling the Agent

How the Agent traverses the environment can affect its capability a significant amount - as a result I first implemented these movements individually before pursuing the joint environment.

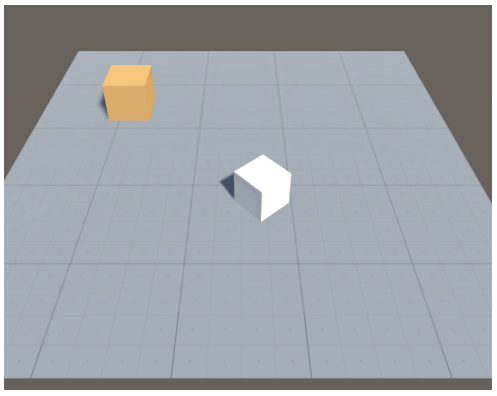
3.1 RollerBall

The Ball Agent was implemented in the ML-Agents Tutorial[2] so it made for a good starting point. After training for a short amount of time the AI made quick work of the control scheme and could navigate easily to the goal.



3.2 Tank Controls

Tank controls are infamous in the world of games for humans, as it is notoriously difficult to navigate and acclimatise to. I attributed this mostly to human error and thought the AI should be able to navigate it just as easily as the ball. It took longer to get the hang of it than the ball but traverses fairly easily.



4 Training the Agents

Training the Agents is naturally very time consuming and quite trial-and-error oriented. As discovered, the Agents often don't react as expected. Additionally, there are a lot of variables that can contribute to how they may act after training.

4.1 Multi-Agent Training

The first hurdle to tackle was figuring out how to train multiple Agents alongside each other. The typical way of training an Agent and the way used in the tutorial[3] can only train one at a time - which hinders this environment a lot. There is a type of training however, called Curriculum Learning, which allows Multi-Agent training.

Curriculum Learning helps Agents learn with a learning curve that steadily increases with each success. So each time the Agent reaches its goal - the Academy changes the environment to make the goal more difficult to achieve. In my environment, this is not something I need or want to implement. However because Curriculum Learning is the only type of training that allows for multiple Agents to learn together - this is the process I need to use. It requires a curriculum to be written as a .json file that specifies what changes and how often. Because I don't need anything to change, this is my .json file. More on Curriculum Learning can be found on the ML-Agents Github[4].

```
{
    "measure" : "reward",
    "thresholds" : [200, 300, 500],
    "min_lesson_length": 1000,
    "signal_smoothing" : false,
    "parameters" :
    {
        "Param-0" : [0.0, 0.1, 0.4, 0.6]
    }
}
```

4.2 Final Agents

The final trained models I ended up with are interesting to observe. The ball works mostly as intended but the ramp Agent has been an issue each version I attempt to train.

The ball model has the following reward conditions:

```
float myDistanceToTargetOld = 1000;
public override void AgentAction(float[] vectorAction, string textAction)
{
    // Actions, size = 2
    Vector3 controlSignal = Vector3.zero;
    controlSignal.x = vectorAction[0];
    controlSignal.z =  vectorAction[1];
    rb.AddForce(controlSignal * speed);

    // Rewards
    float distanceToTarget = Vector3.Distance(transform.position, target.position);
    float distanceToWedge = Vector3.Distance(transform.position, wd.position);

    // Reached target
    if (distanceToTarget < 1.42f)
    {
        AddReward(5000.0f);
        Done();
    }

    if (distanceToTarget < myDistanceToTargetOld)
    {
        AddReward(1.0f);
        Debug.Log("lessthan" + myDistanceToTargetOld);

    }

    if (distanceToTarget > myDistanceToTargetOld)
    {
        AddReward(-2.0f);
        Debug.Log("greaterthan" + myDistanceToTargetOld);

    }

    if (distanceToWedge < 0.5f)
    {
        AddReward(1.0f);
    }

    if (distanceToWedge < 0.5f && transform.position.y > 1.0f)
    {
        AddReward(2.0f);
    }

    myDistanceToTargetOld = distanceToTarget;

    // Fell off platform
    if (transform.position.y < 0)
    {
        AddReward(-500.0f);
        Done();
    }
```

These statements either add or subtract rewards from the Agents depending on the action they perform. The ball Agent is rewarded for approaching the target and when reaching the target - as well as being rewarded for being on top of the ramp(“wedge” in the code). It is subtracted rewards for moving away from the target and when falling off of the platform.

These conditions should train the ball to move towards the target and when the ramp is near, climb it. Additionally it should train it to not fall off the platform or move away from the target.

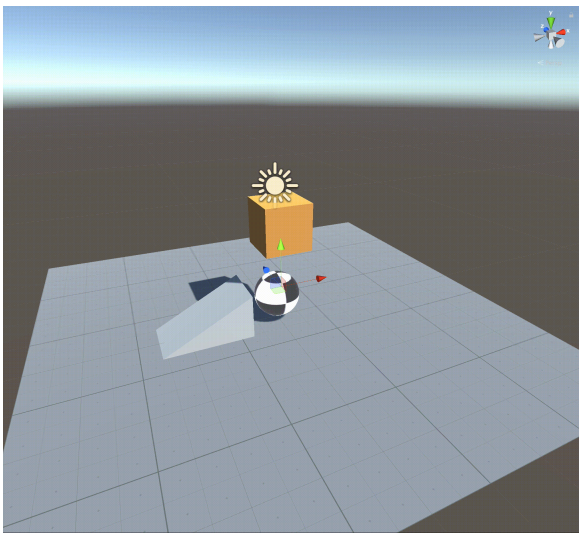
The ramp model has the following reward conditions:

```
public override void AgentAction(float[] vectorAction, string textAction)
{
   MoveAgent(vectorAction);


    // Rewards


    float myDistanceToTarget = Vector3.Distance(transform.position, sp.position);


    if (myDistanceToTarget < myDistanceToTargetOld)
    {
        AddReward(1.0f);
        Debug.Log("lessthan" + myDistanceToTargetOld);

    }

    if (myDistanceToTarget > myDistanceToTargetOld)
    {
        AddReward(-2.0f);
        Debug.Log("greaterthan" + myDistanceToTargetOld);

    }


     myDistanceToTargetOld = myDistanceToTarget;
     Debug.Log(myDistanceToTargetOld);


    // Fell off platform

    if (transform.position.y < -0.1)
    {
        AddReward(-500.0f);
        Done();
    }

}
```

The ramp Agent is rewarded for approaching the ball Agent and penalized for moving away from it, as well as falling off of the platform.

Given these conditions, you would expect the ramp Agent to follow the ball Agent and try to get as close to it as possible. In theory, with these two trained Agents, the ball would approach the target, the ramp would approach the ball, and once the ramp is close to the ball - the ball would roll up the ramp and hit the target.

Strangely, this is not what happened. The ball Agent acted as expected, and approached the floating target, waiting under it - as close as it can get to the target without the ramp. However, the ramp merely spins in place.



I have tried many variations of these two Agents together with varying results - however the ramp repeatedly misbehaved. I believe it may be the tank control scheme that could stop it moving as it would like: though it seems to traverse fine when left alone to meet the target(3.2). In other tests it has also been able to traverse as normal so perhaps the spinning is not a result of this, though it is the most suspicious component.

5 Moving Forward

I believe the main reason this experiment failed was the time constraints I had. I think most issues encountered can be worked around and I have some ideas for what may work. Due to the nature of a project like this ideas can have setbacks and though in theory it may work there can be problems you don't expect. Given more time I believe my goal could be achieved through these ideas.

4.2 What Might Work

Though technically less co-operative, I believe a psudo-co-operative AI can be achieved without Curriculum Learning. Here are the steps:

- Set up the RollerBall scene WITHOUT the Ramp
- Train the ball to go towards the goal as per usual
- Set up the ramp object and raise the goal so the ball cannot reach it without the ramp
- Run the pre-trained ball model in the scene while the ramp AI trains

This would require a long amount of training as the ball is unaware of the ramp, and the ball wouldn't be a co-operative Agent. However the ramp AI, being a co-operative Agent, would serve to help the ball reach the goal despite the ball being unaware.

Another idea that is completely untested but may yield some results, is training one AI while having a human player control the other AI. This would teach one AI at a time to co-operate with a human. This way they would not be trained to co-operate with each other

but they would have some understanding of co-operation within the scene and may be able to reach their goal when put in the scene together. This could also potentially yield better results for an Agent that is meant to co-operate with humans. A setback however is that training takes a long time and a human player would need to be playing the entire duration.

4.3 Other Co-operative Ideas

Some of the other co-operative concepts listed above(2.2) are interesting and I believe can yield good results. Using Curriculum Learning to train two Agents at once(or perhaps using the AI/human training idea stated above(4.2)), I believe the walking on hinges concept could work well. The main reason this wasn't pursued is due to time constraints and getting the physics hinges working well. Given the time to create this mechanic, I believe the two AI would be able to walk to their common goal. The Agent would need to be aware of:

- The position of the goal
- The position of the other Agent
- The position of itself
- The rotation of the other Agent
- The rotation of itself

The easiest co-operative environment would be the two Agents pushing one heavy block to a common goal. This would only require one type of trained Agent, and neither would need to be aware of the other Agent's position. The scene would simply be:

- A target area
- A heavy block(weighted so it can only move if two Agents are pushing it)
- Two smaller blocks (the Agents inhabit, a lighter weight but enough to push the big block together)
- Both blocks will have the same AI trained model

This AI can be trained by simply having the training environment only include one AI block and by making the heavy block light enough to be pushed to the goal by one AI. This would train the AI to push the block to the goal. When put in the new environment with the heavy block, each AI would approach the heavy block and try to push it endlessly. Once both AI are pushing it, it would be able to be moved towards the goal. This is technically co-operative but only just - as they are not even aware of each other.

5 Conclusion

Though there were many setbacks that stopped this experiment from creating a perfectly co-operative Agent, I believe it is fully achievable. Given more time and tweaking these environments could succeed in training their appropriate AI. I have never seen a trained co-operative Agent through ML-Agents and, given a successful experiment, achieving this could streamline a lot of AI creation within game development. I would love for myself or somebody else to go forward with this project and see it out to a successful end.

Bibliography:

[1]https://github.com/Unity-Technologies/ml-agents
[2]https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Create-New.md
[3]https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-ML-Agents.md
[4]https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Curriculum-Learning.md