- AlphaZero is a RL algorithm developed by Google DeepMind.
- It is a general AI that can play a wide range of board games, and is currently the reigning world champion in both chess and Go.
- It can be thought of as a robust model predictive control algorithm

- If you did 3F3 we learnt that MPC uses a receding horizon to look ahead and return its next action
- in this way, AlphaZero is very similar except it performs a directed tree search to return a control action, rather than optimising over all future states in a receding horizon

- Episodes are played by taking turns between a player and an adversary, both of which are trained via reinforcement learning.
- This massively improves on MPC because the player takes into account what it thinks an optimal adversary would do.
- The adversary can be thought of as a series of optimally adverse disturbances, or the "worst case scenario"
- Which theoretically trains the controller to be robust against any disturbances that it faces

- Other potential benefits:
- Non-Linear Systems as tree search will work with any deterministic system
- Easily applied to many dynamical systems as long as you know the dynamics and have enough computing power.

- The faster the control inputs, the better they approximate continuous control.
- So theoretically AlphaZero can be made into a Robust to disturbances, pseudo-continuous MPC algorithm

- What AlphaZero is trying to create is a function that will take some board state as input and output a very high quality move that we can make.
- There are two clever algorithms that AlphaZero uses:

- The first is a neural network, which represents the AI's intuition of how good a move is. It output's:
    - A pmf over actions, that represents how good it thinks the move is
    - A value, which represents the expected game outcome where -1 = certain loss and +1 = certain win, with all the values in between.
- Once you have this, the obvious next step is to just do an argmax – take what looks like the best move.
- But we can do better, this leads to the second algorithm

- The second algorithm is a MCTS, which drastically improves the p.m.f. over the actions that we initially guessed, and also the expected value
- If we look ahead and apply the neural network again and again, alternating between the adversary and player's turns.
- there might be some move that looked good initially, but now you've seen what the adversary will do and what you would then do
- A different move looks better

- This directed tree search dramatically reduces computational time since we don't have to do an exhaustive search, we just look at the most likely paths.
- For example in it's match against the previous reigning chess AI, stockfish, it only looked at 20,000 possible branches, whereas stockfish looked at 3bn per move played

- We can then take this improved pmf and value for the state and retrain the neural network on these
- In this way, we can improve through playing another agent
- And then we can do this again and again continually improving our policy/neural network

- As a test-bed the inverted pendulum was chosen because it has some good qualities: Inherently unstable, highly non-linear, dynamically simple
- The player was chosen to act with just +1, -1 at F_1
- The adversary was chosen to act +1, -1 at F_2.

- The cost of each state is a measure of how far away from the unstable equilibrium the IP is
- Conversely to the player, the adversary tries to increase the cost

- The adversary was chosen to act at F_2 so that two distinct policies are learnt – rather than one just outputting inverse probabilities of the other
- There are two neural networks, one for the adversary and one for the player.
- They take turns with the state and output an expected value and an action pmf.

- With the IP the episodes can last indefinitely and there is no easily discernible winner – therefore cant use expected outcome for the value
- The true value is the DISCOUNTED FUTURE COSTS and is calculated after an episode is completed as it needs to know the future state

- One of the most difficult questions was how powerful the adversary should be.
- The adversary has the advantage of gravity and it's actions are more powerful for the same force.
- We can work out roughly how much more powerful it is by finding what force from each agent gives the same angular acceleration of the pendulum
- However due to non-linearities and differing accelerations at the base, this does not make them equal. In fact making them equal is almost impossible

- Adding a "roughly equal" adversary initially causes the pendulum to fall instantly, which leads to no training examples.
- Therefore a much weaker adversary must initially be used.

- The solution tried was pretraining the player without an adversary, and then introducing a handicapped adversary.
- The adversary must always be weaker – or have less of an effect on the system than the player.

- This is so that the player has at least some chance of balancing the pendulum when fully trained.
- Theoretically this shouldn't be an issue for the adversary as it should be able to capitalise on errors that the player makes.
- For example, using a traditional optimal controller it is easy to push the pendulum over by applying force at the natural frequency of the pendulum, resonating it into falling.

- This is what happens when the player, pushing along the bottom, has had no training and is relying on just the monte-carlo tree search to look ahead to determine the moves it makes
- The MCTS is much better than playing at random
- But even with no adversary the player has a hard time balancing the pendulum if it's not trained

- Doing this, and then adding an adversary and training it as well gives something like this
- It starts off well, sort of resting the pendulum against the adversary.
- But the cost is constant for this – which the player doesn't like, so it tries to move towards the middle to reduce this
- Which it achieves, but then finds that it cannot balance against the adversarial weight in the middle
- So it drifts to the right, slowly increasing its distance from the centre and therefore the cost
- And in this way, the adversary wins slowly

- This is the main problem with using AlphaZero for a control application.
- The worst case adversary isn't necessarily the same as the opposite of the best action.
- For the IP, the adversary learnt a long-term strategy since it could not beat the player by simply over powering it.

- ….
- Needless to say, the player having an internal model of an adversary that only pushes in one direction does not generalise well to random adversaries.
- With a random adversary, every time it pushes the way in which the player is not expecting, the tree essentially resets, making the tree only as deep
- As the average time between when the adversary makes an unexpected move
- This means that once a trained adversary is introduced, the performance of the controller is pretty much constant.

- One possible way to overcome this “over training” of the adversary would be to make the system more realistic
- For example adding a first order lag may allow the adversary to take advantage of the resonance of the system and push it over.
- However, there was not enough time to include this