
There are No Weight Classes in Game Dev:
Competing Alongside Goliaths
Game AI North
@tduringer
TEMPLE GATES
There are No Weight Classes in Game Development: Competing Alongside Goliaths
www.TempleGatesGames.com

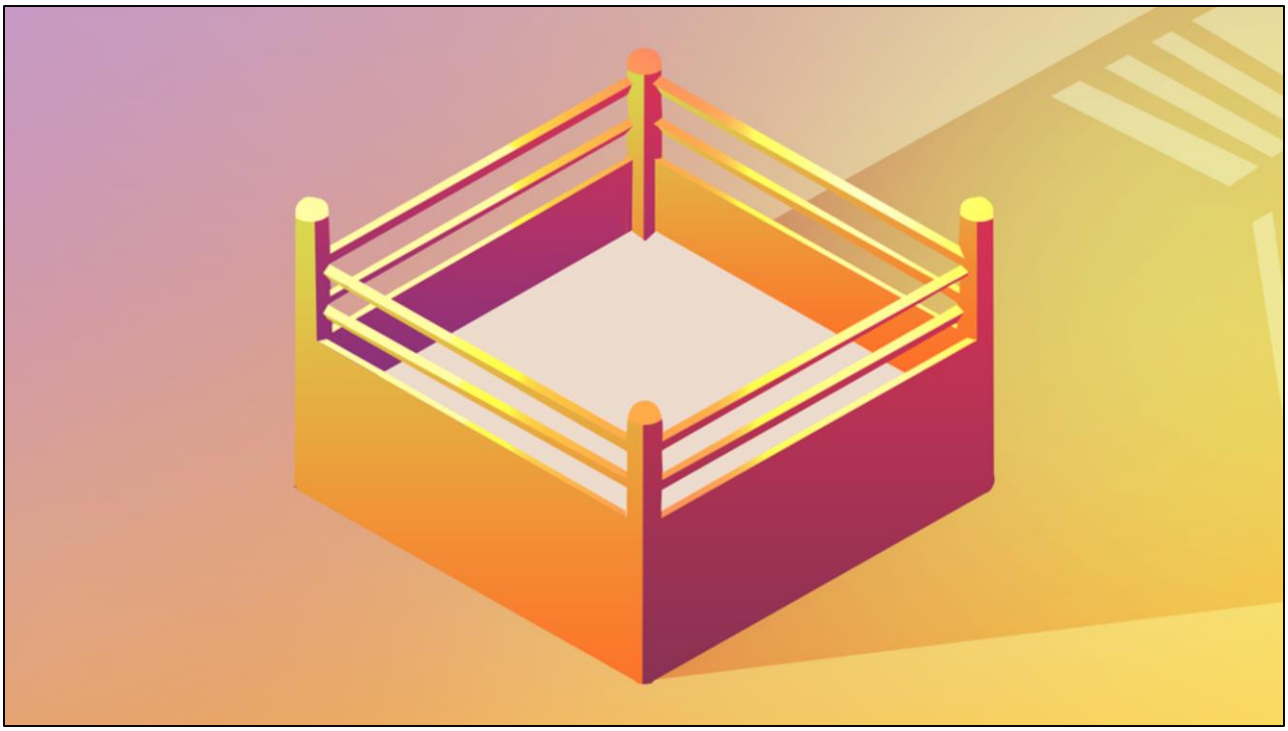
Title

No weight classes

There are no weight classes in game development. There's no division to keep the fights fair. You, your team, or your studio are going to be competing with games from vastly different team sizes, with vastly different budgets. BUT you can use AI, UI, and performance tricks to make your games stand out among those from Goliath teams.

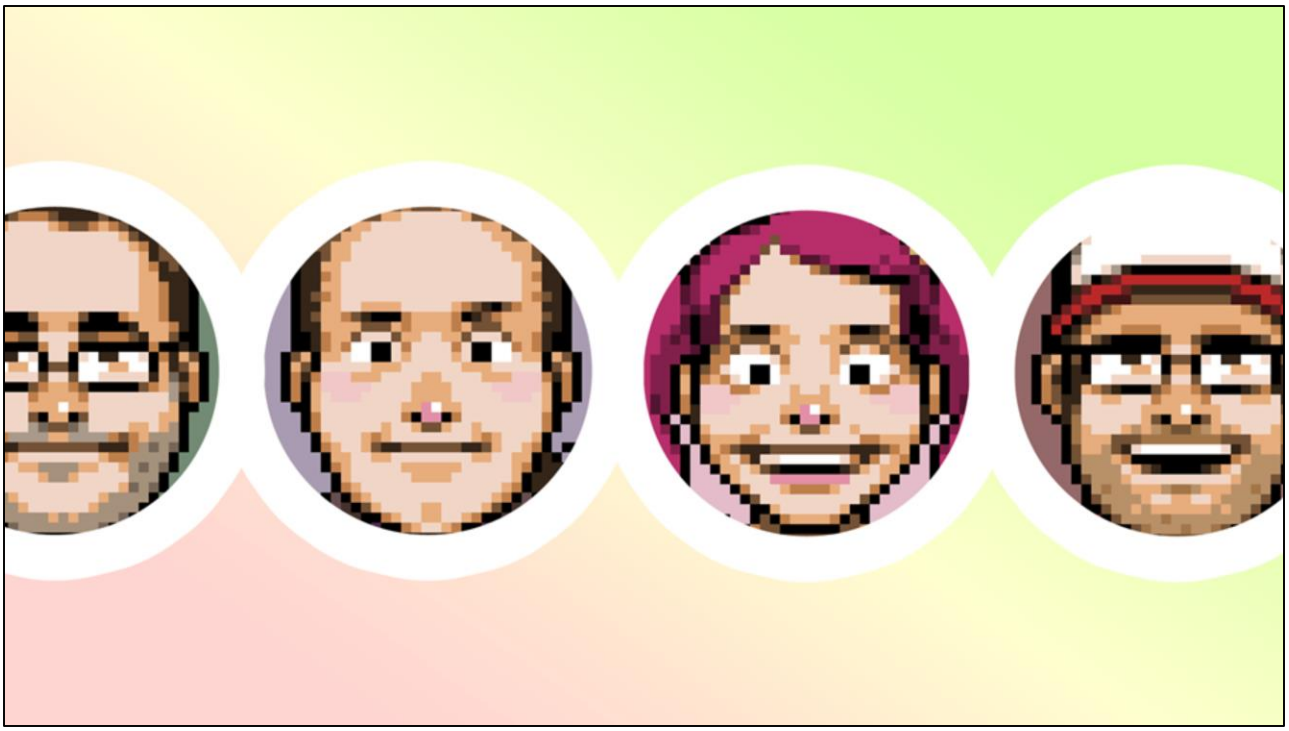
Intro Race

I’m going to talk to you about how we did this in Race for the Galaxy, because that’s my most recent project, but these lessons apply across a lot of games - although not all of them, and I’ll get into which ones and why.



Innovations

So, in Race for the Galaxy, our neural network creates its own training data with which to hone itself. We run our AIs on a separate thread, to keep framerate up while it's executing. We conduct the flow of AI events through locking attention tokens to make a complex system intuitive. A few smart innovations can give a small team the edge to stand out in the ring. And I'll talk through each one of these and how to do it, and if it would make sense for you.



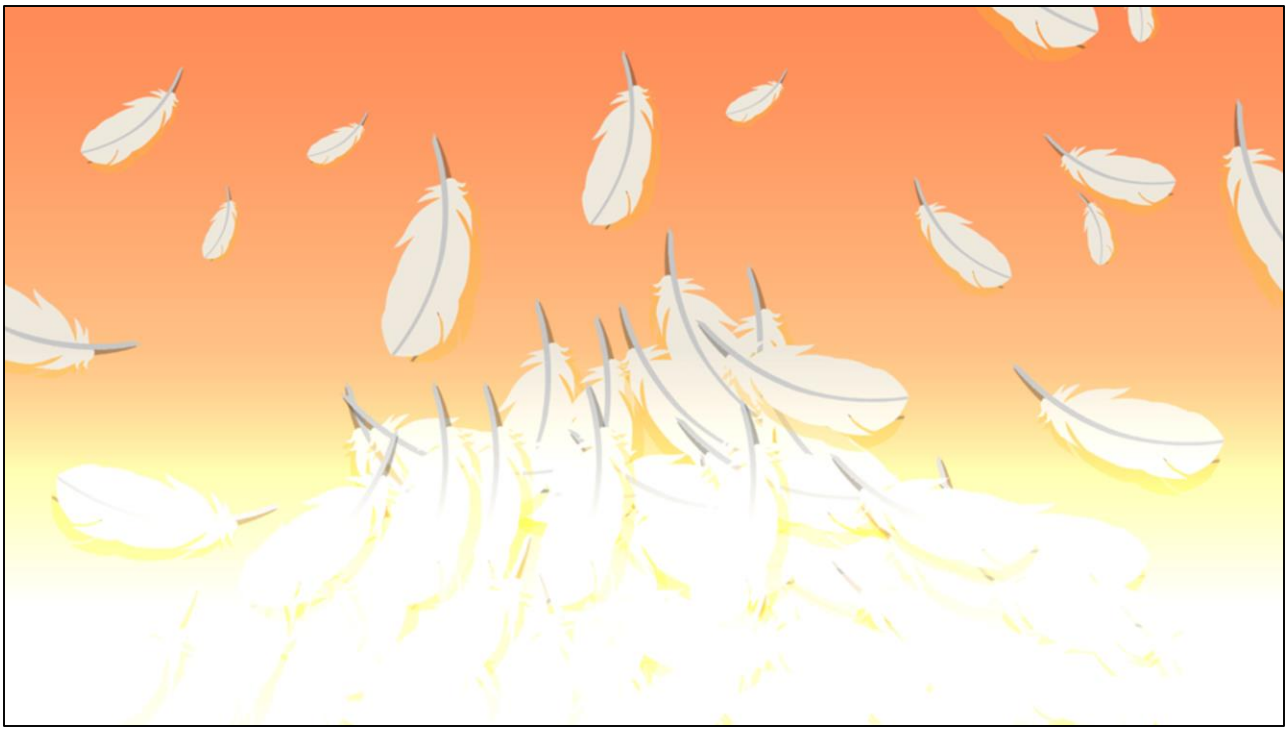
Studio History

But first, let me give you a little background on why we made Race in the first place. This is us. We're small! Just four people.

Featherweight
Teams like this are the featherweights. Maybe some of you are too? And that means we're up against middle weights and heavyweights in the ring…

Games Shelf

Our games are being sold next to theirs. The consumer doesn't know one is made with millions and one is made by three friends. How can we possibly compete?



… not to mention thousands of other featherweights. I'm sure you've heard that 40% of all games ever released were on Steam in 2016. It's easy to get lost in the shuffle.

Niche

But because you're small you can go niche. When you're small, your budget is small, the time investment is short, and you don't need to sell a million copies. You don't need to be mainstream. At first, we chose VR.

Experimenting

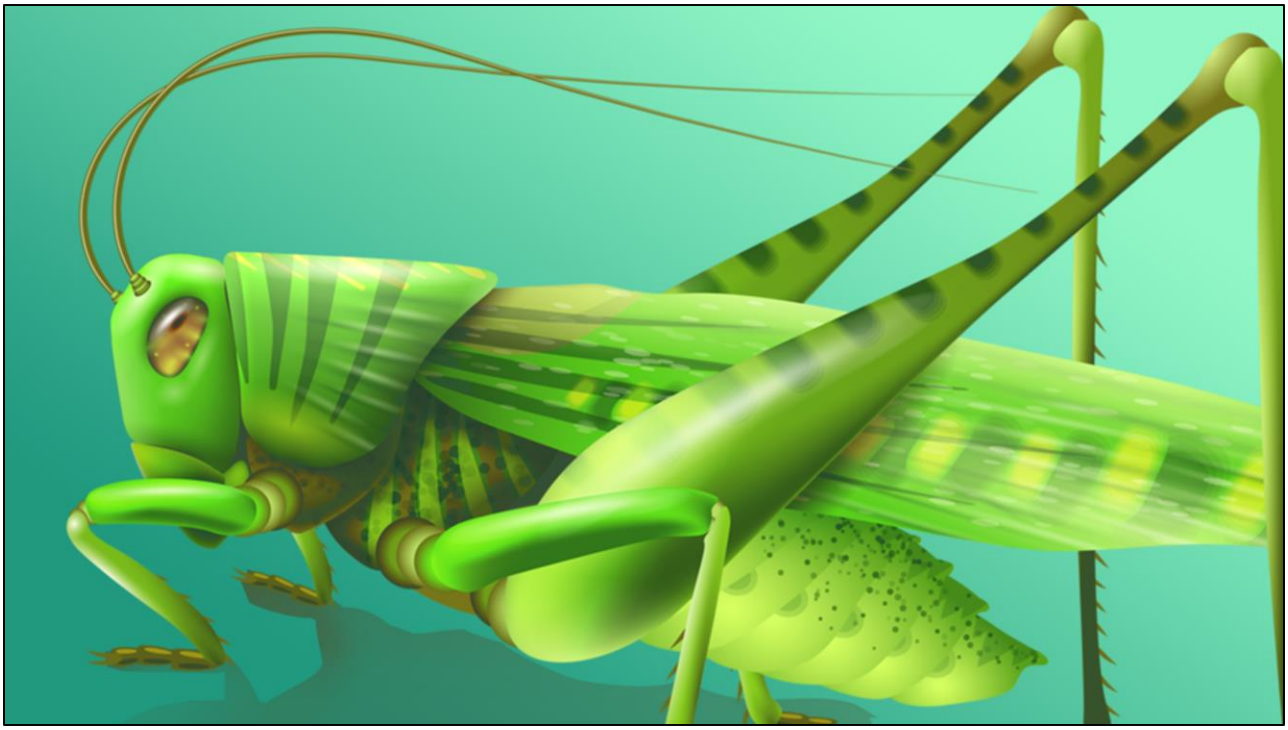
We chose vr. Our team had been experimenting with Gear, Vive and Rift.

Bazaar

We created this game Bazaar. A flying carpet game. It was really cool, basically a roguelike puzzle labyrinth game. It was procedurally generated and fun to explore. But there was one problem, it was lonely. When you're on a rollercoaster, you want to ride with a friend, look over at them, and share the thrill. This was a single player game.

Mutliplayer lobbies

So in our next game we brought players together, but we wanted to give them something to do, an activity. So we made one of our favorite boardgames, Ascension in VR. A new niche. Ascension is a deckbuilding card game. And it was really cool! You could inhabit the characters from the game. You could play with people from around the world, and I did. I've met people I consider some of my closest friends in this game.

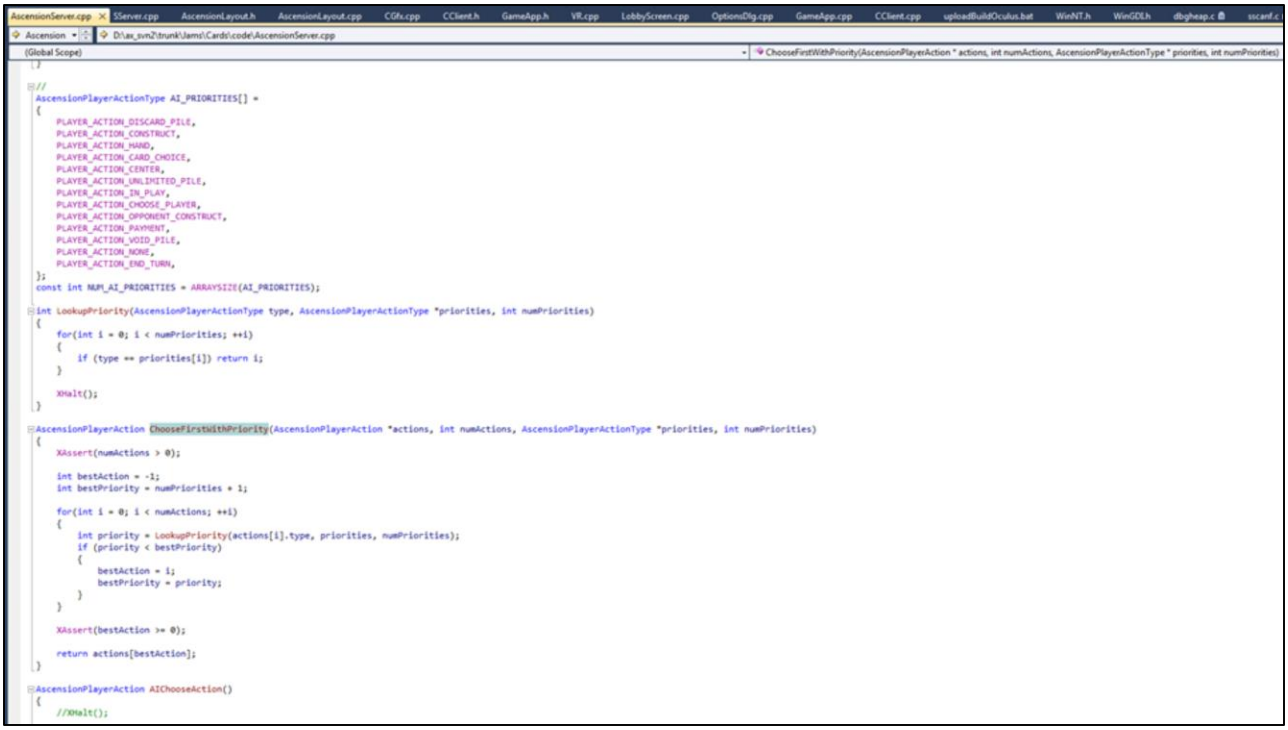


Crickets AI

But we were having a problem. We didn't always have enough players. The dreaded Crickets. There are a few reasons for that. The VR market is immature. Headsets are prohibitively expensive. Not to mention you need a cutting edge PC to run VR. All this amounts to friction for players getting into the game, so we needed to develop an AI to offer single player mode.

AI comes to life

And In VR, your AI opponent needs to come to life. Animation wise, it means an AI that responds to the game. It taps its foot when you're taking a long time. It does frustration animations when you pull off a combo. It plays victory animations. The point of this is to communicate high level state changes in the game and give visual feedback to the player. You're doing well. You've erred.



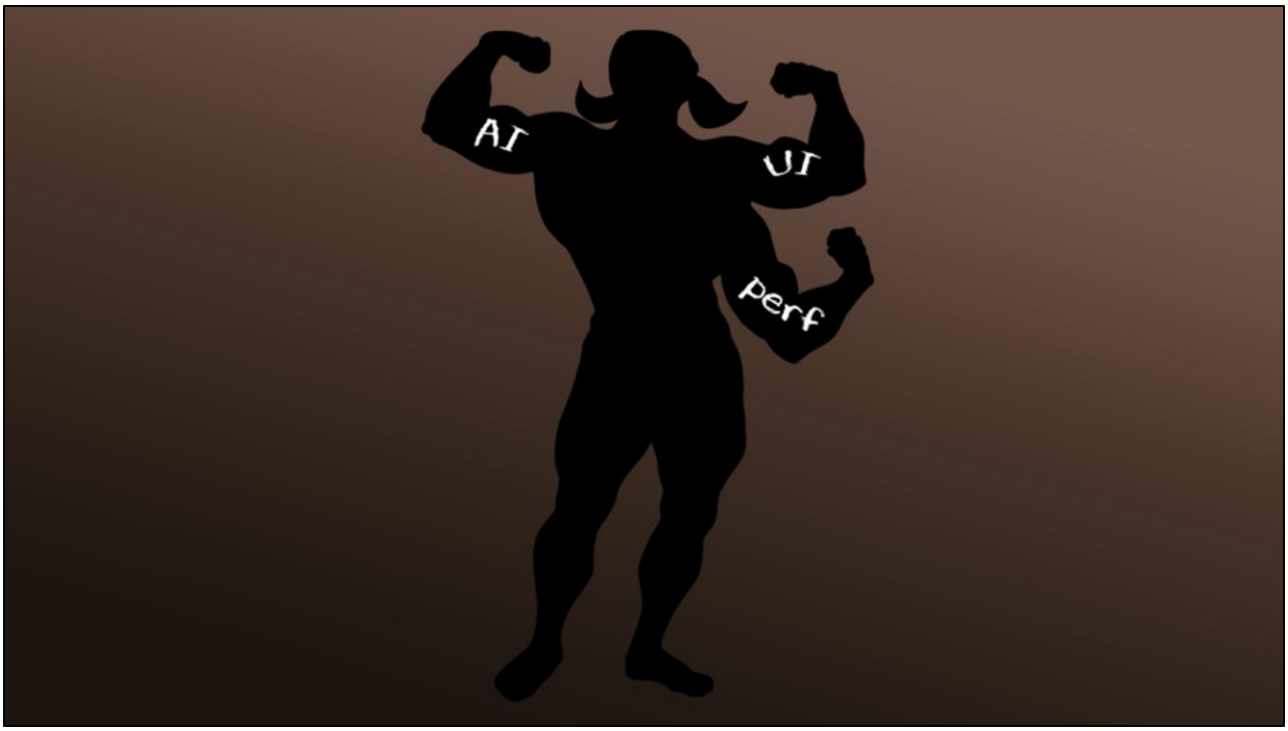
Heuristic AI

But under the hood strategically it's heuristic driven AI. This is an AI based on expert plays based from exemplars. For Ascension, this meant an AI that favors cards with a high VP to cost ratio. If you know the game that will make sense to you, I don't need to explain the whole board game. It buys center cards before resorting to Heavies or Mystics. It plays all its cards first before buying. When destroying an opponent's constructs, it chooses the highest cost one. It trashes militias and apprentices first. All of these heuristics are defined by human experts. It's the things that we think are the best playa. And it's pretty good. But I'll talk about how to make your board game AI even better.

Strengths

When you're figuring out what to make next, you want to double down on your strengths. For us AI, UI, and performance, were areas of investment.

AI
UI

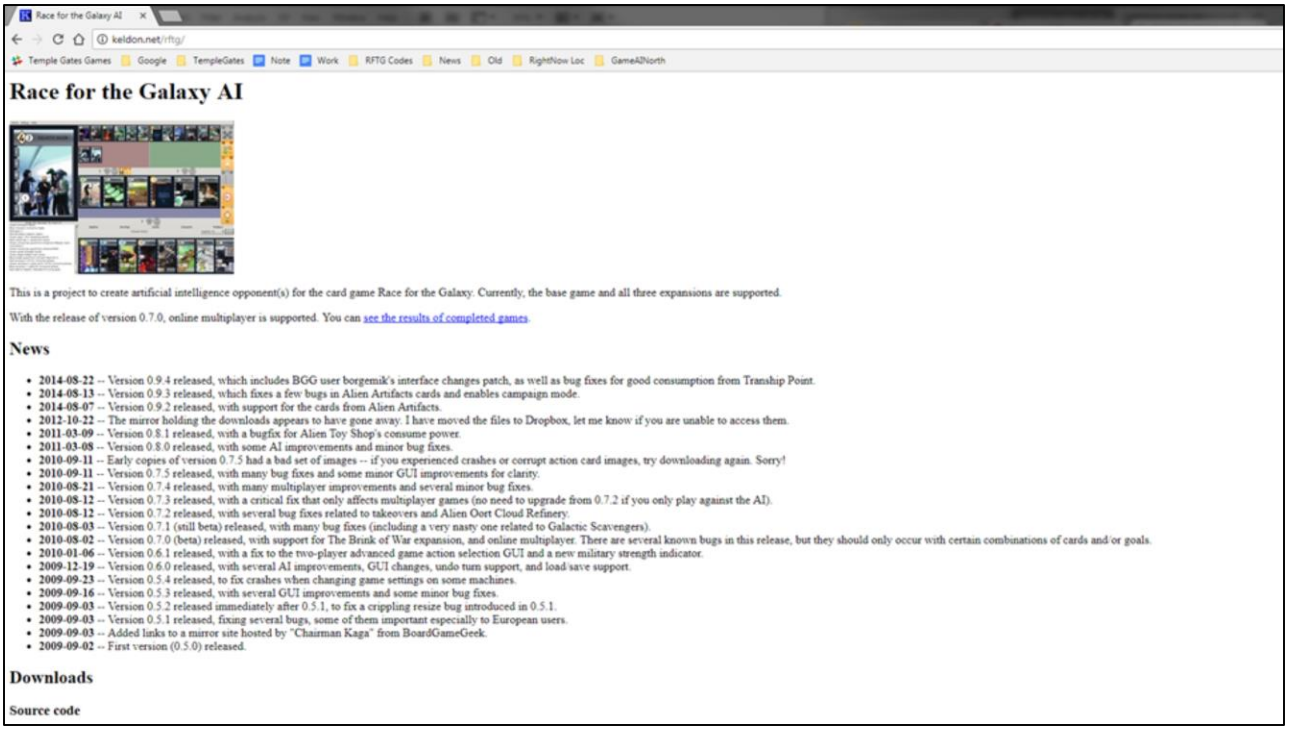

AI
UI
perf

Tackle Race

So we decided to tackle Race for the Galaxy. An engine building cardgame, set in a sci fi universe.

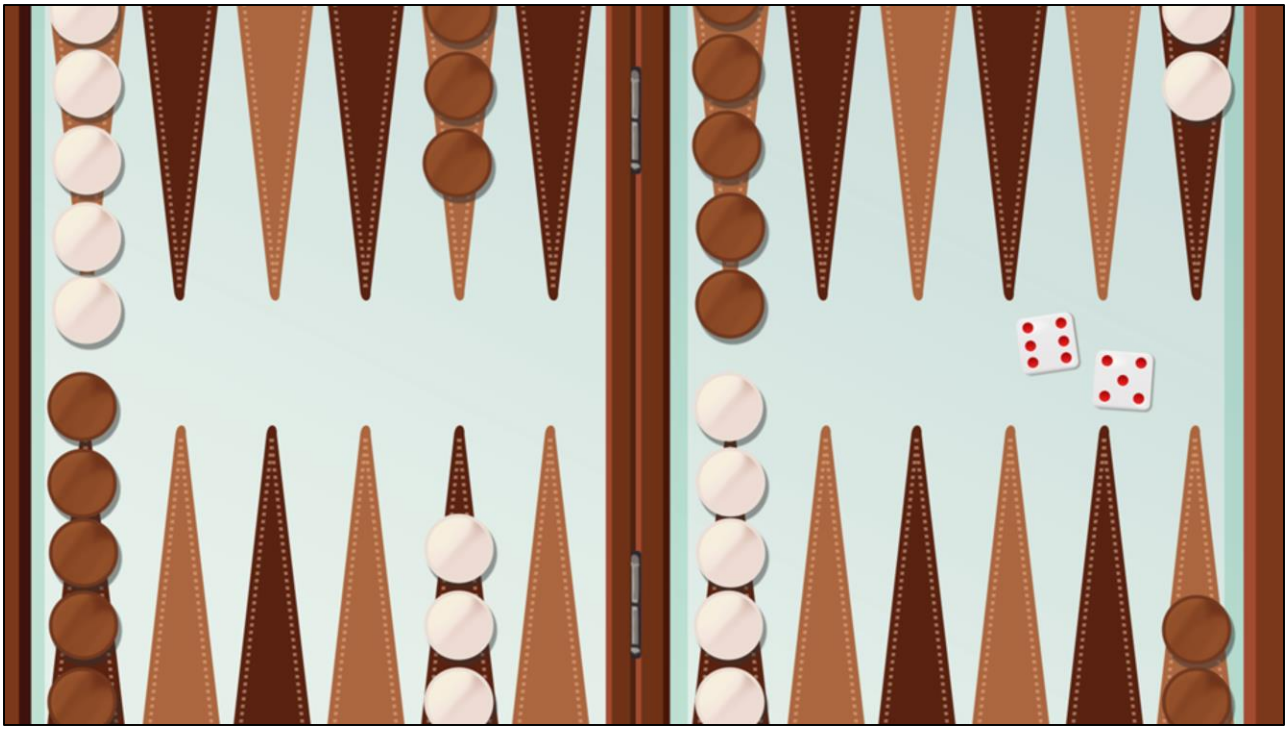
Neural Network

In Race, our AI takes a different approach from Ascension, which is heuristic driven. The Race AI is much more challenging to the user because it's powered by a neural network.



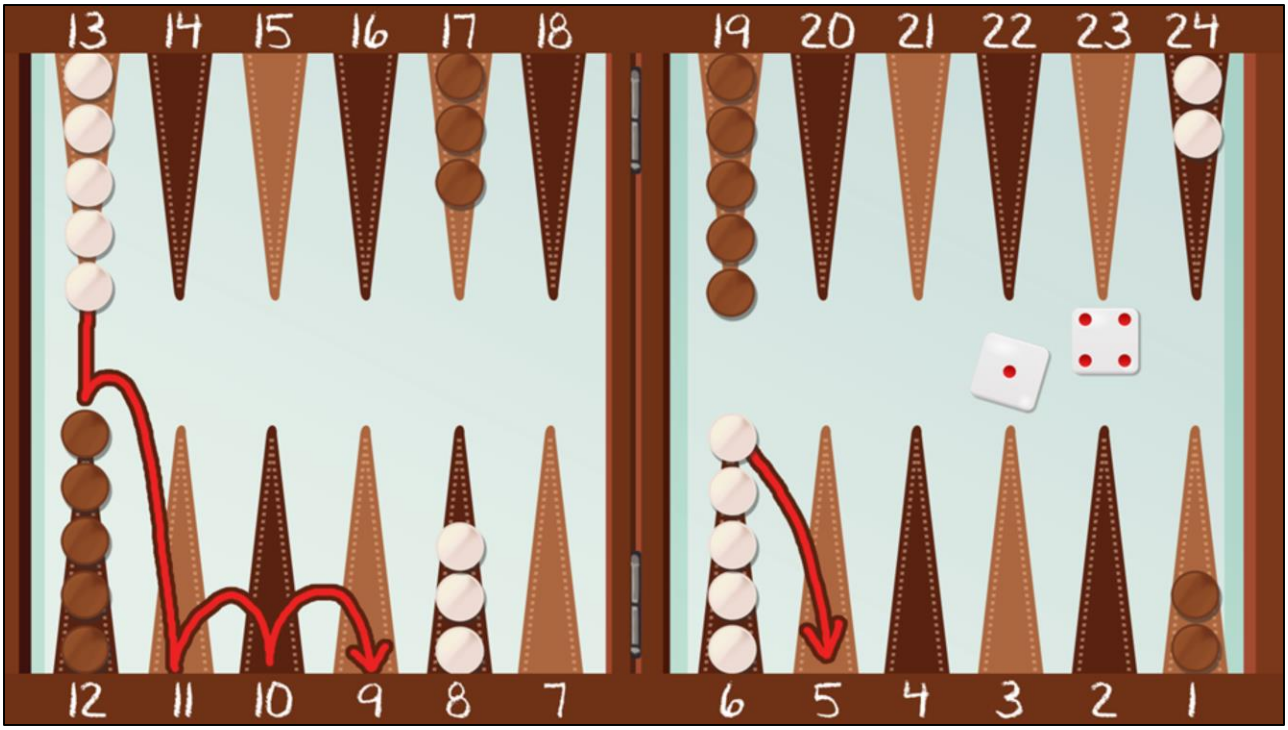
Keldon 2009

In fact this neural network was originally developed by Keldon Jones, and was first published in 2009. That's a while ago! Neural networks have blown up in the last 5 years especially in fields of natural language processing, computer vision, autocorrect, delivering ad content. But before it was all the hype, Keldon was creating one of the first practical applications of a neural network for gaming.



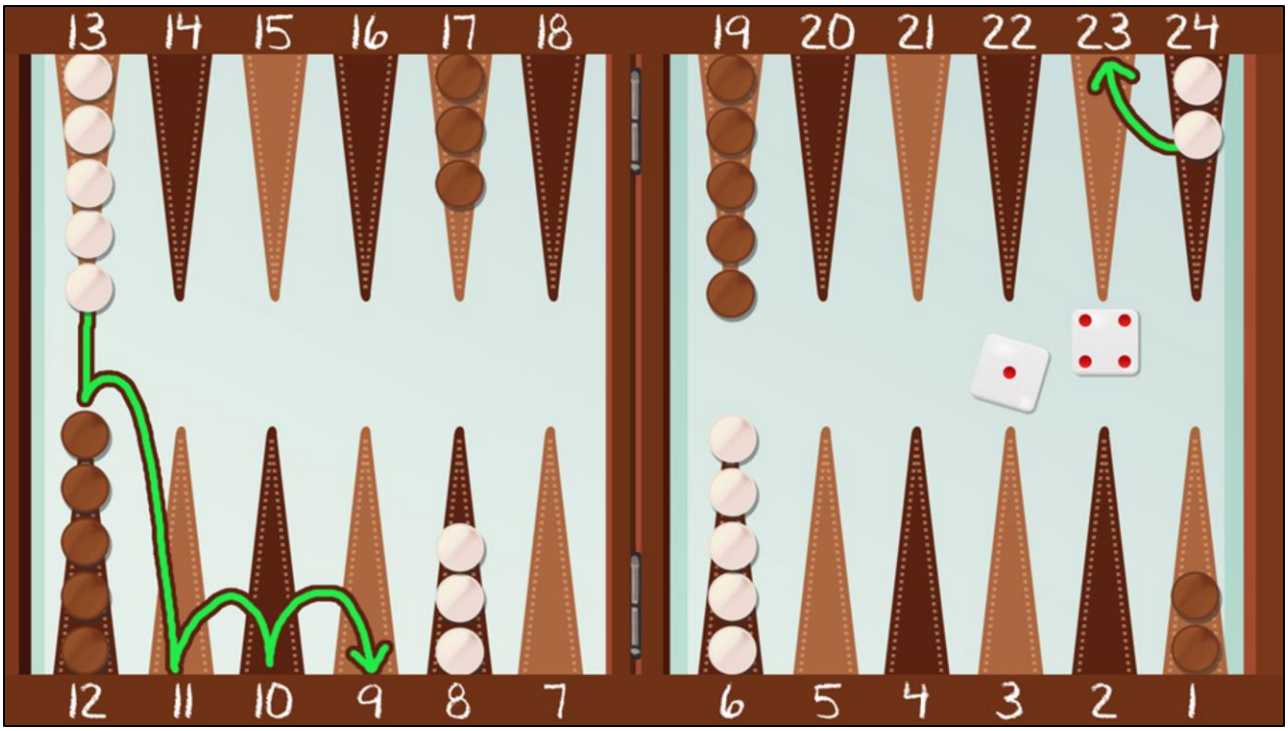
TD Gammon Intro

And this was based on the original research for TD Gammon by Gerald Tesauro. I want to talk to you about one of the most cutting edge game AIs ever created, based on a 5000 year old game: Backgammon. Tesauro pioneered an AI that ultimately changed conventional wisdom (the culmination of 5000 years of humans playing) on how BackGammon should be played. This AI successfully developed on its own unorthodox strategies that were eventually adopted at the highest tournament levels and are now widely accepted by players around the world.



Slotting

For example, on the opening play, if 4 - 1 dice rolled, traditionally, players “slotted,” they move from 6 to 5. What this play is doing is saying ok, if brown rolls a 4, or a 3-1, or a 2-2, it will be hit, but there is a small chance the white checker in slot 5 won’t be hit, in which case white can have a very strong position in space 5. This was accepted as the go-to play.



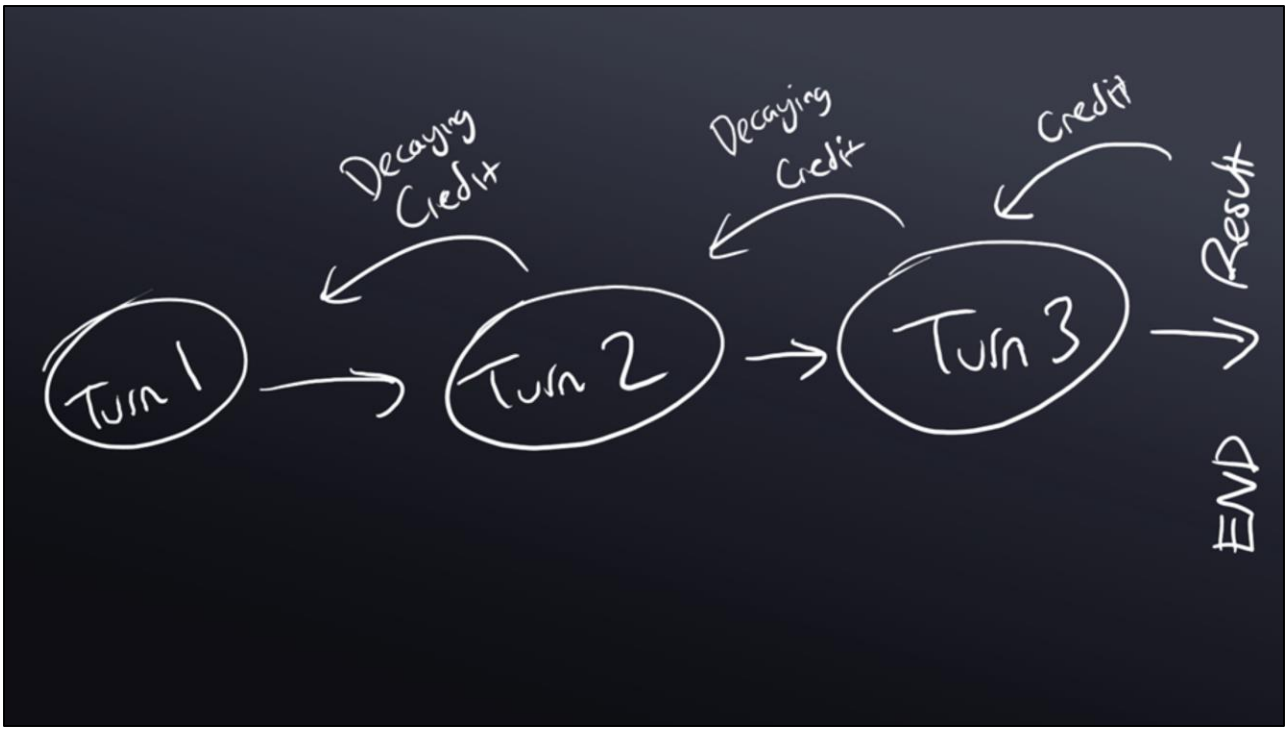
Splitting

But TD Gammon, based on a neural network, found higher success rates over thousands of games by “Splitting” rather than “slotting”, and moving one checker off of 24 to 23.

	Move	Estimate	Rollout
Convention	13-9, 6-5	-0.014	-0.040
AI	13-9, 24-23	+0.005	+0.005

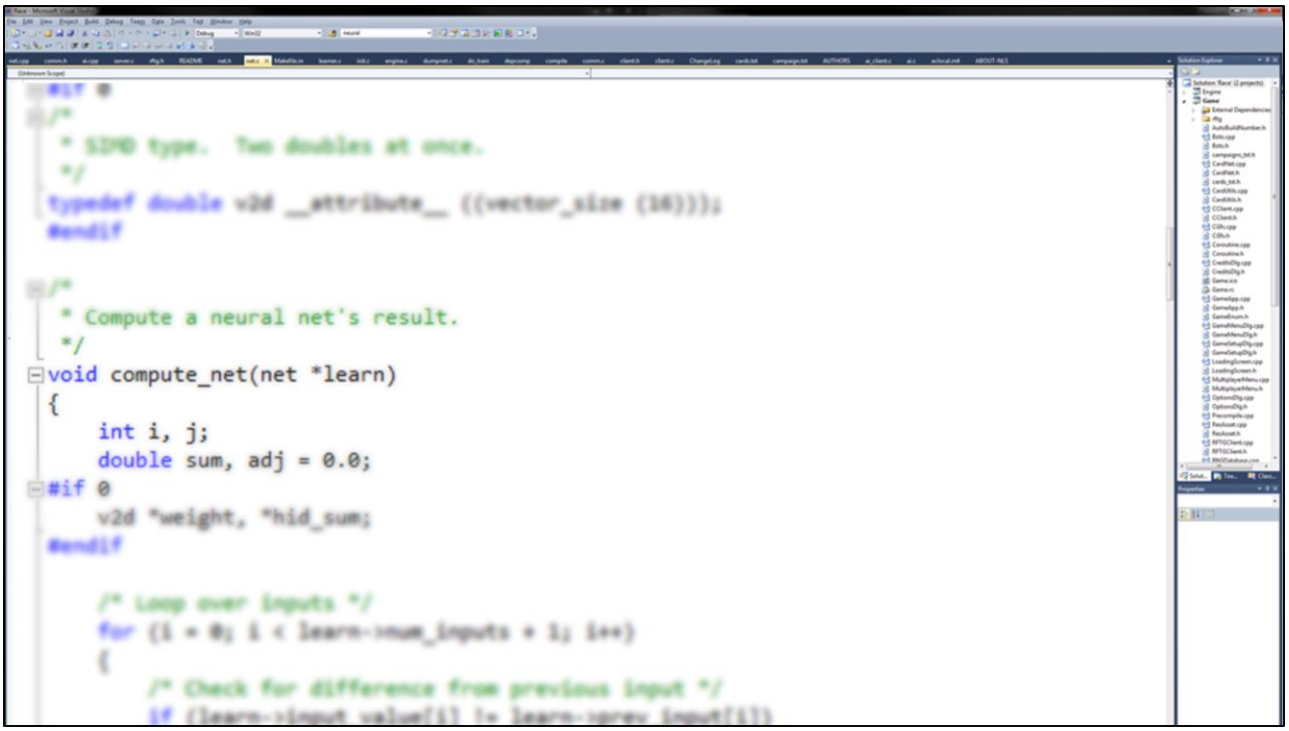
Novel Strategy

It's a better strategy with higher win yield. How did this AI discover a strategy that humans hadn't in 5000 years? The AI starts with zero initial knowledge. So unlike our heuristic driven AI in Ascension, it doesn't come pre populated with imperatives based on human expertise, in essence, it's teacherless. And that's good, because human expertise being emulated is not infallible.



Temporal Credit Assignment

What it uses is reinforcement learning based neural network where the AI learns input, produces an output, and receives a reward based on feedback signal. The goal is to choose the best action for optimal reward, but the reward at the end of the game is delayed, so a temporary credit or blame is assigned at each turn leading up to the final reward at game end. In a Temporal Difference method, learning is based on temporally successive predictions, and I'll get more into that.



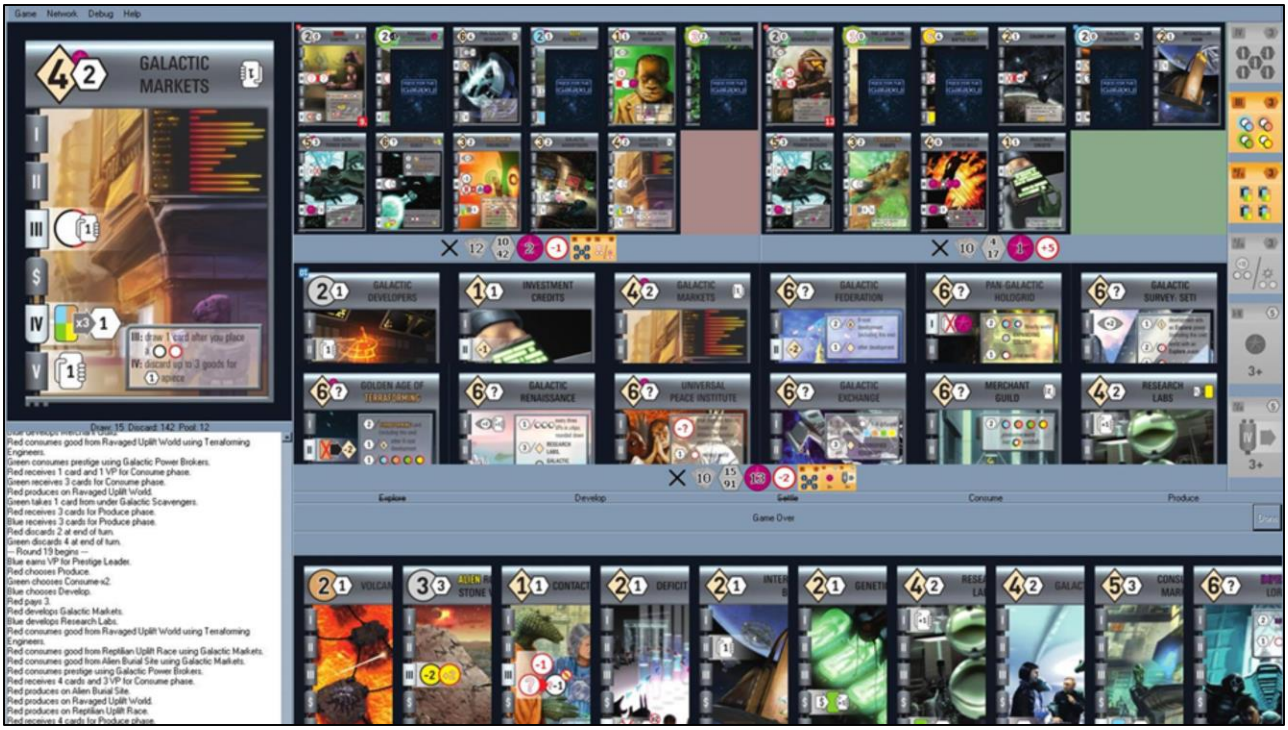
```
#if 0
/*
 * SIMD type.  Two doubles at once.
 */
typedef double v2d __attribute__ ((vector_size (16)));
#endif

/*
 * Compute a neural net's result.
 */
void compute_net(net *learn)
{
    int i, j;
    double sum, adj = 0.0;
#if 0
    v2d *weight, *hid_sum;
#endif

    /* Loop over inputs */
    for (i = 0; i < learn->num_inputs + 1; i++)
    {
        /* Check for difference from previous input */
        if (learn->input_value[i] != learn->prev_input[i])
```

Keldon's code

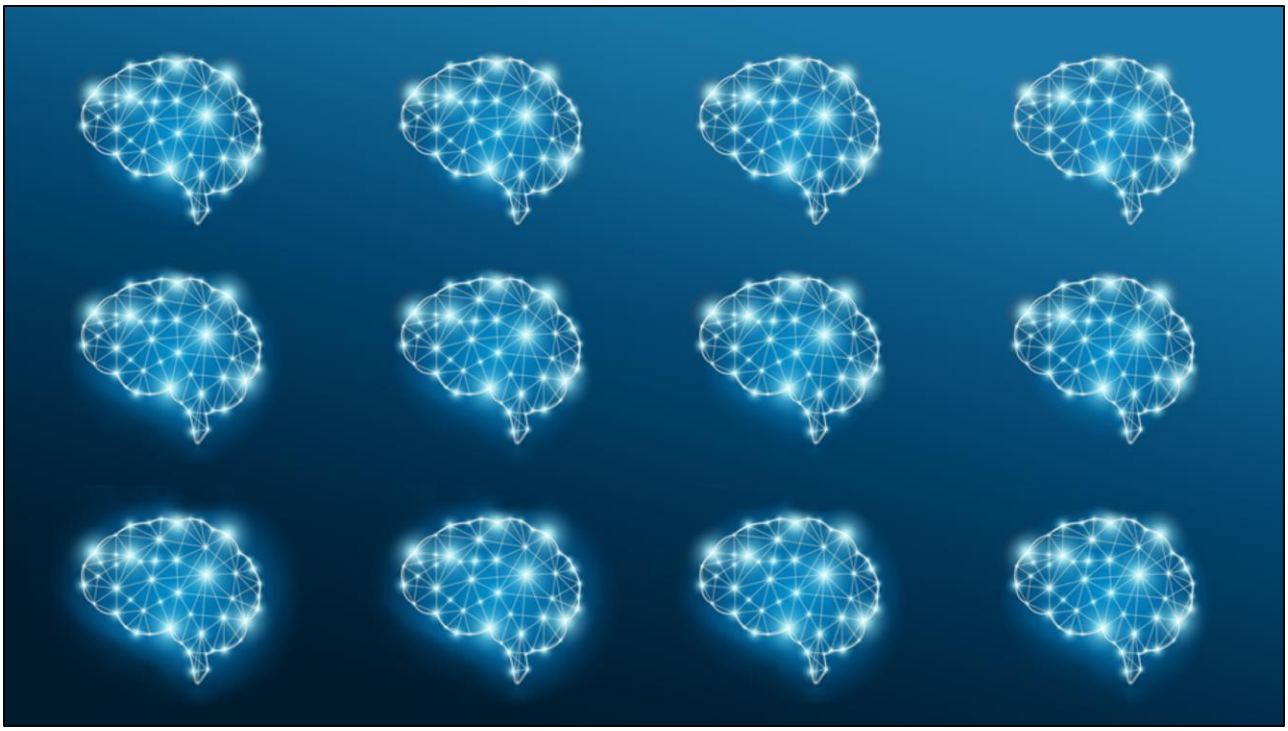
Keldon built several neural network AIs using this Temporal Difference method. You can download the code yourself and start poking under the hood, in fact, that's what I did!



Keldon Client

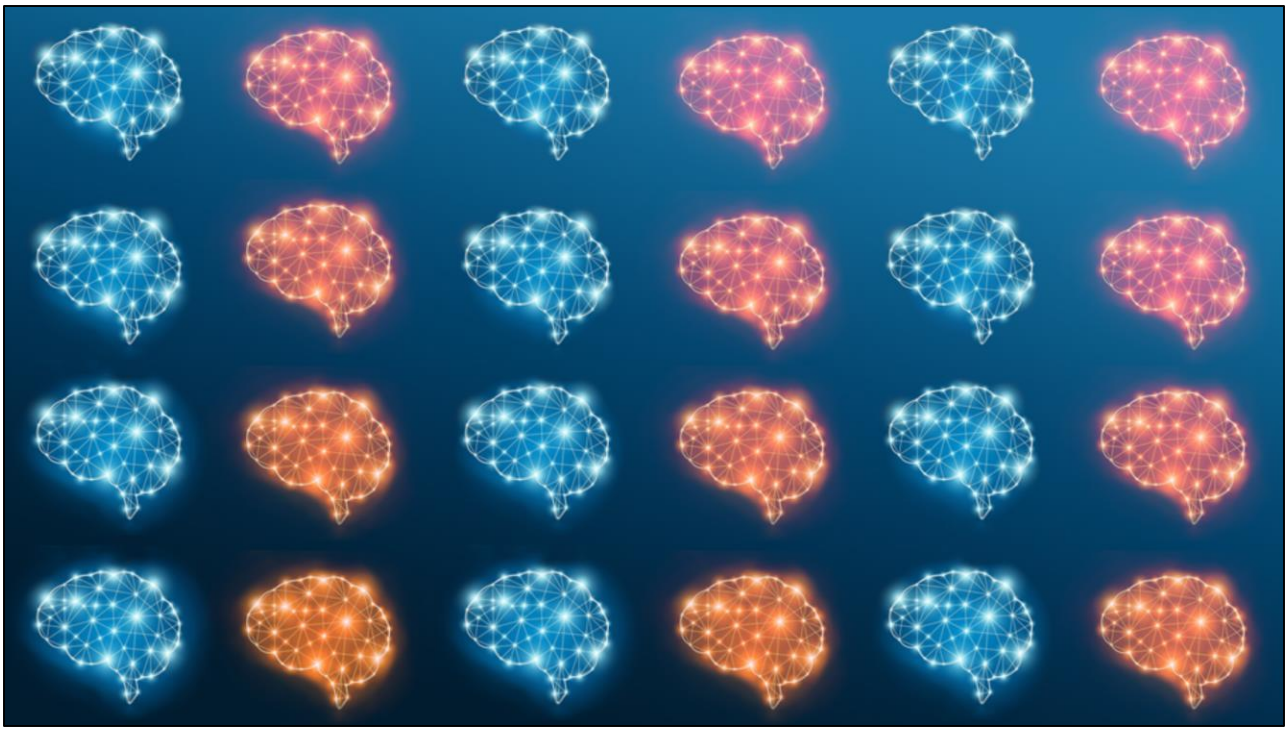
I was tinkering with this code because was looking to see if there was a digital version of my favorite board game - Race - and I found one! It was an unlicensed open source project developed by Keldon. I loved it, and I wanted just one thing to be added to it: a feature to play a sound when an opponent had taken a turn, because you can play this against others from around the world, while you surf the web or do other work. If someone's got analysis paralysis and you shift focus to another application, I wanted a notification for when I should pay attention again. That got me thinking about Race. And feature creep city, I thought about all the things I wanted, first and foremost, I wanted it on my phone.

I got in touch with Tom Lehmann, the designer of the game, to see if that was something I could make happen. Turns out he was local! I brought cookies. We played games (we still play games!). And we became friends. People ask how we get the IPs for some of these boardgames, and I think the trick is to actually really really care! It's easy to have deep strategic knowledge of a game when you love to play it every day. Race is my favorite board game. Tom had one request, though. He suggested that I look up this guy Keldon, supposedly, he had a pretty neat AI. An AI I've been following for years. OK!



12 AI

But actually, Keldon didn't just use one neural network. He bifurcates the model. The most important thing is deciding how to architect your bifurcations. We have twelve unique models of neural networks each trained for a different set of expansions and player count. If you're running a two player game, the AI is on a different network than a three player game. We could always partition further, but there are diminishing returns and you're burdened with complexity and size - which is bad. The NNs are one quarter our download size, which is pretty nutso given the volume of art we ship with from all the game cards.

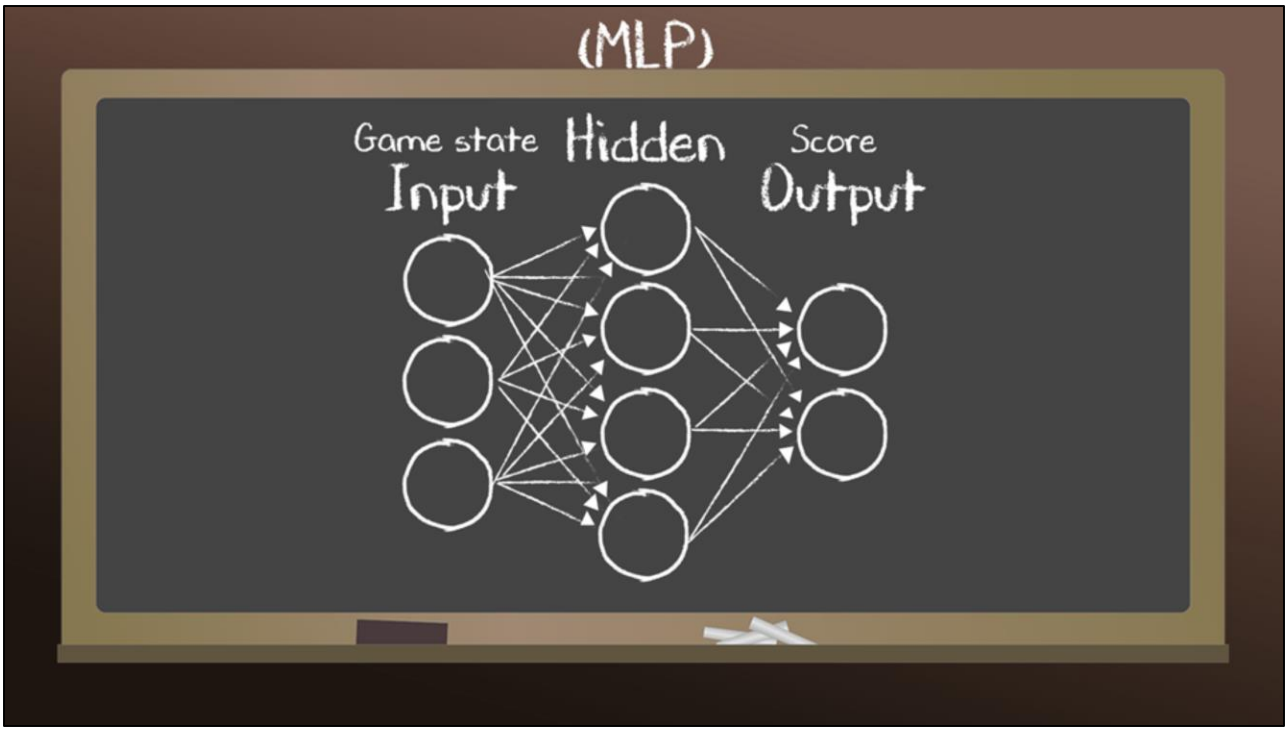


24 AI

For each birfurcation there are actually two flavors of neural networks at work, each with it's own main function. The functions are What move would player X make in state Y? We call that function Eval_Role. The second function is given this board, score it: tell me how good it would be for me on a scale of -1 to 1. We call this Eval_Board.

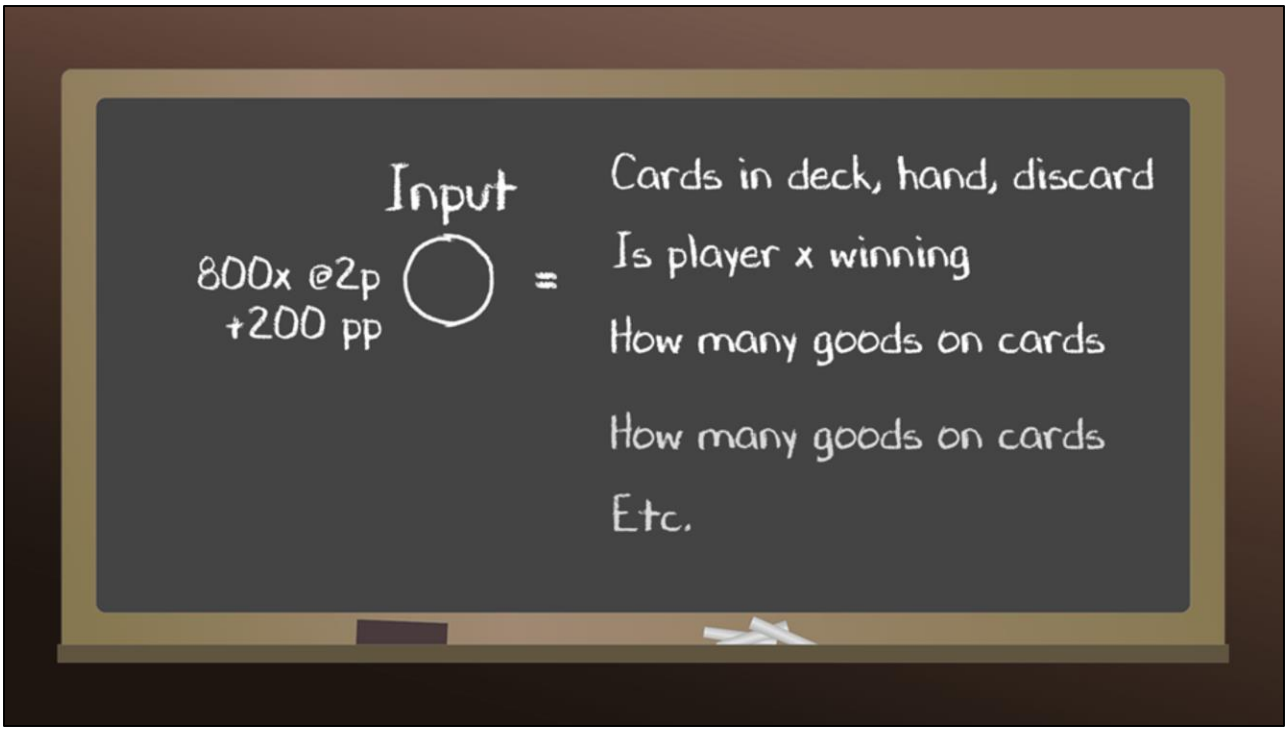
Simulate

On its turn the AI simulates through every possible move it could make, and it runs the Eval_Board on the results and chooses the best one. In fact, to move the simulation forward a step could involve one or more calls to Eval_Role to guess opponents role choices.



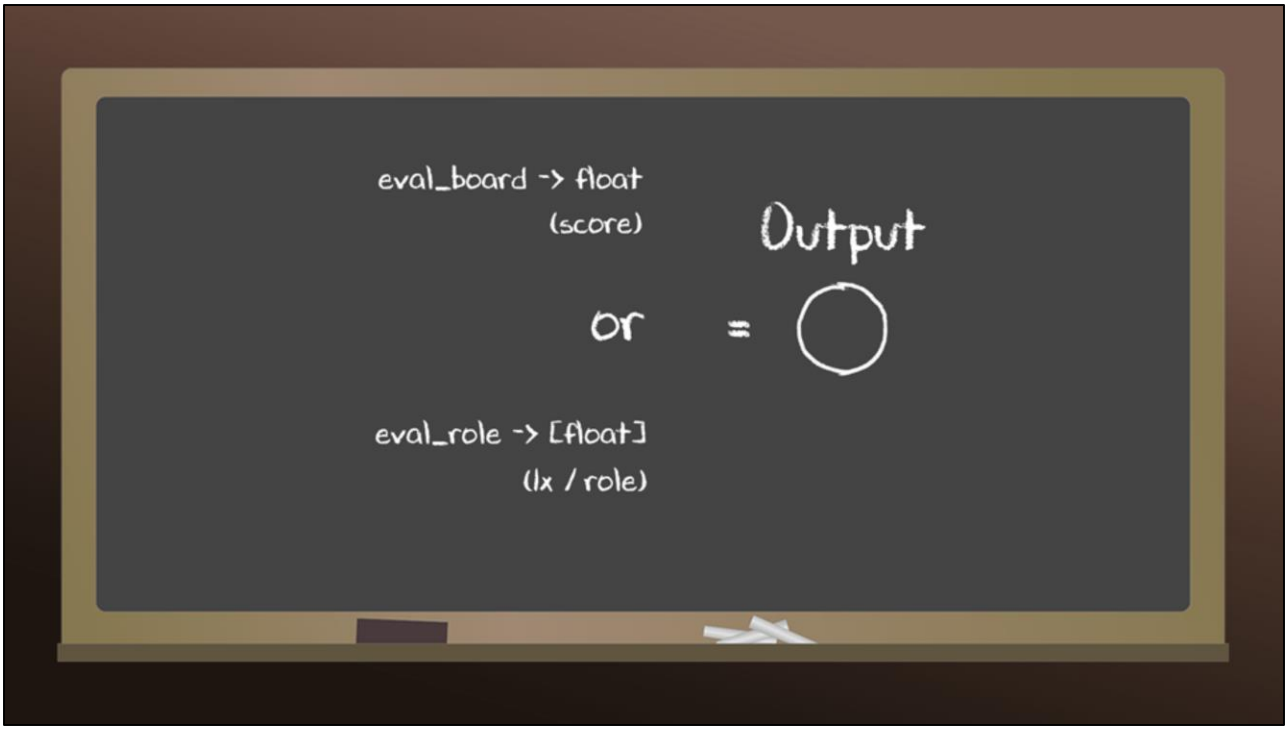
Game State - > Score

This uses a pretty standard Multilayer Perceptron Architecture. You have your inputs, hidden nodes, and outputs. There are weights assigned along edges. The computation is the product of the node and the previous weight into a sigmoid function (we used hyperbolic tangent) which skews the input and output range of the system to be non-linear. This is important because you're just trying to figure out a function, and you don't know if that function is going to be linear or not. Because at the end of the day your Neural Network is basically a black box that is trying to arrive at a function that creates outputs matching your training data.



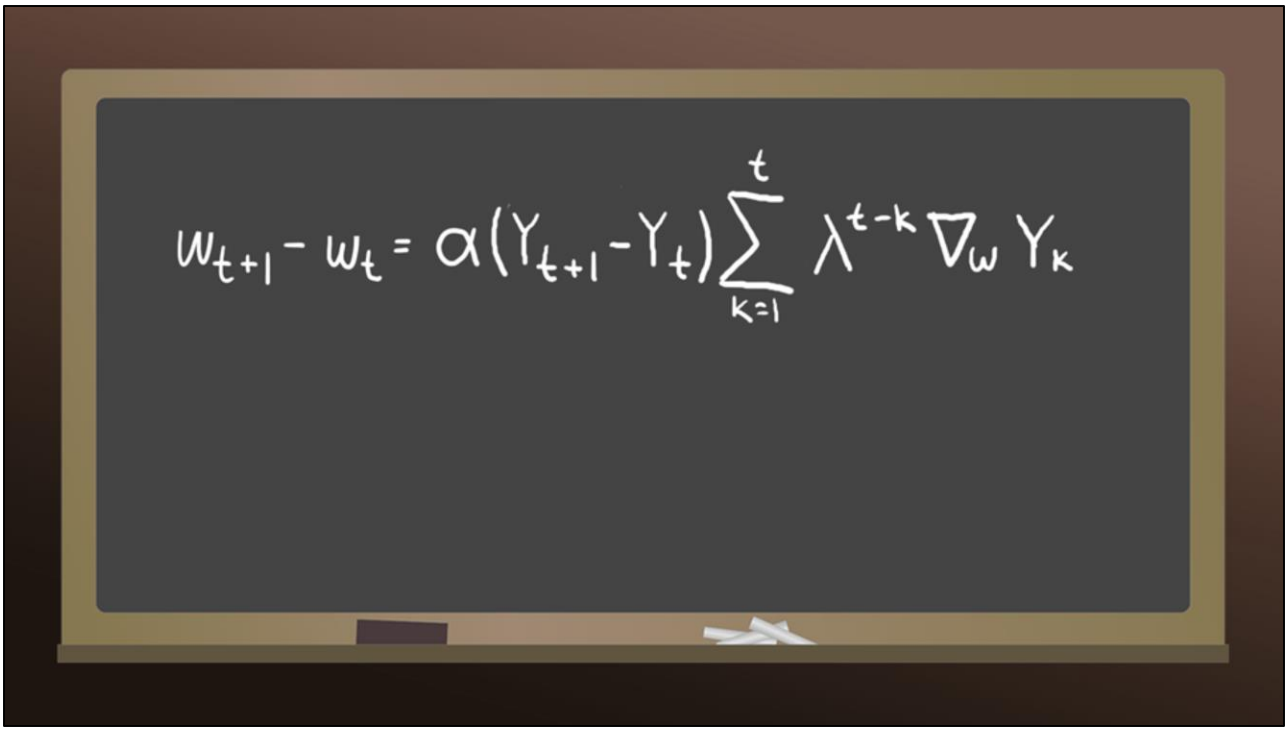
Inputs

In our case, what are the inputs? These are 800 nodes including a ton of game state data.



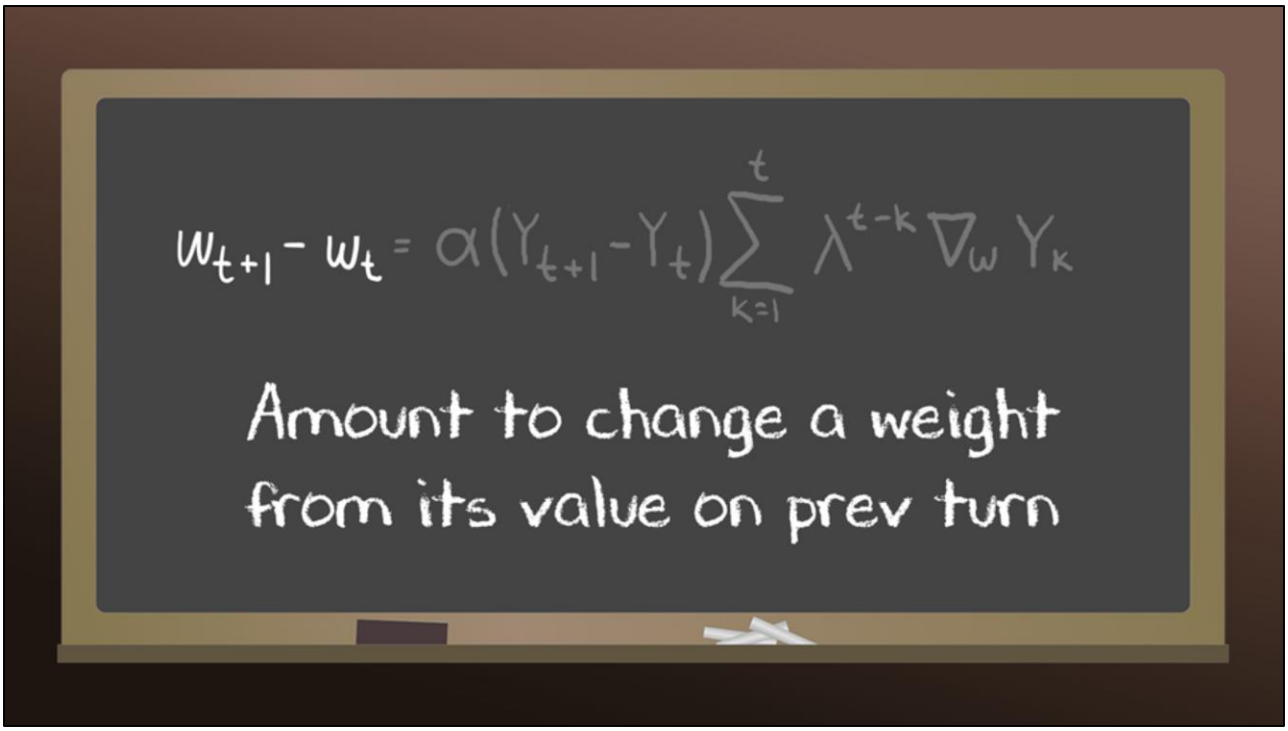
Outputs

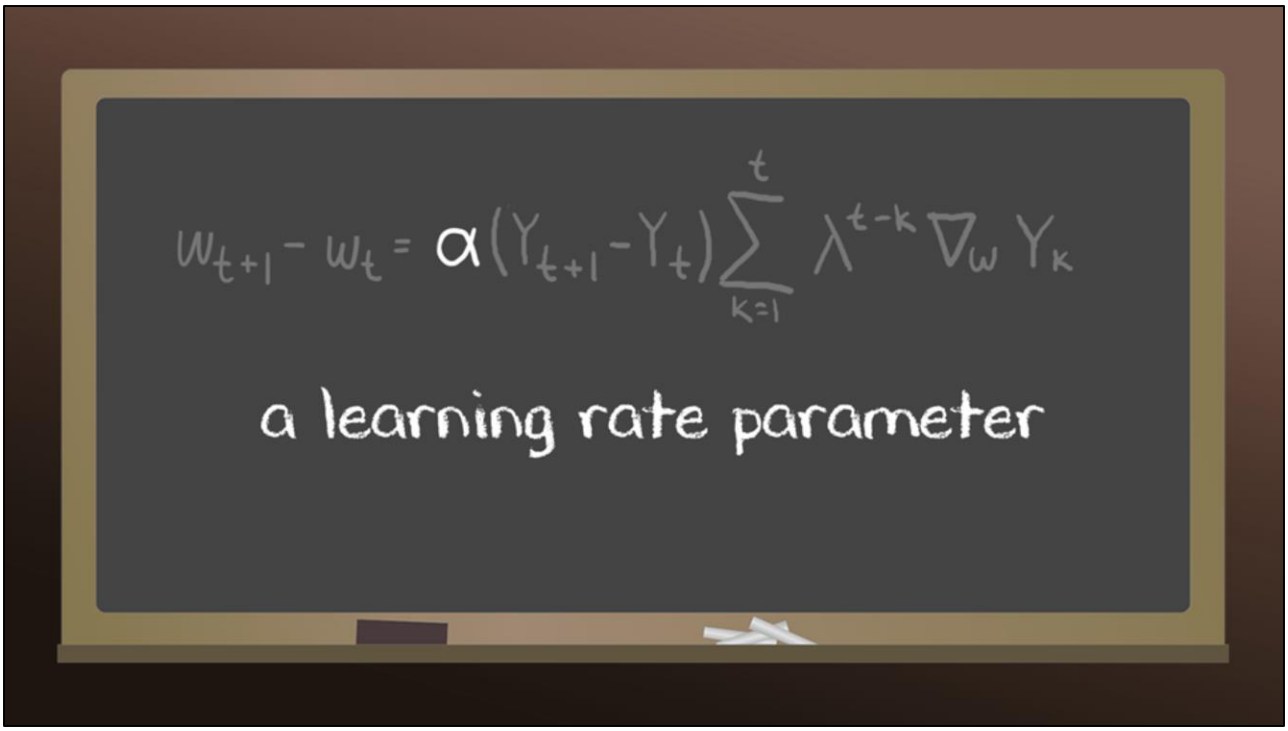
The outputs are calculated through feed forward flow, from input, through hidden node, to output. For Eval_Board the outputs are the win probability for each player. Then in practise we end up just using the win probability for the local player as the 'score' of the board.

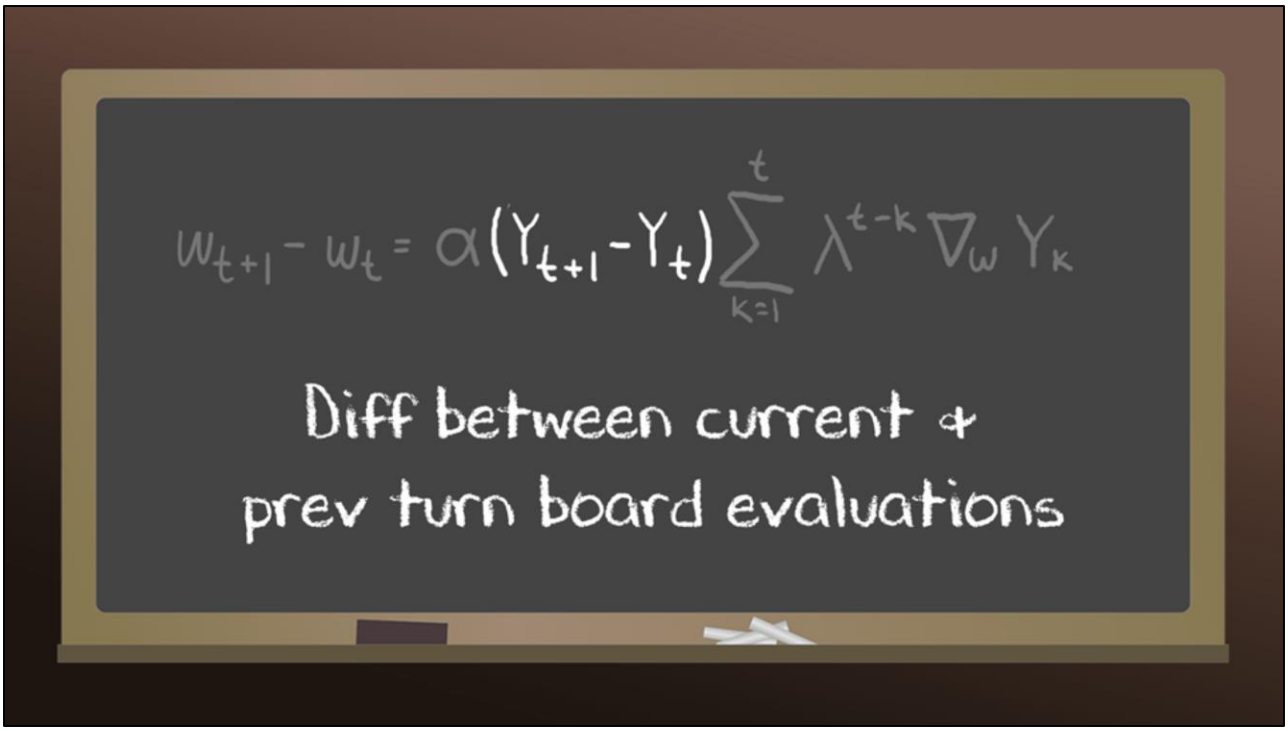


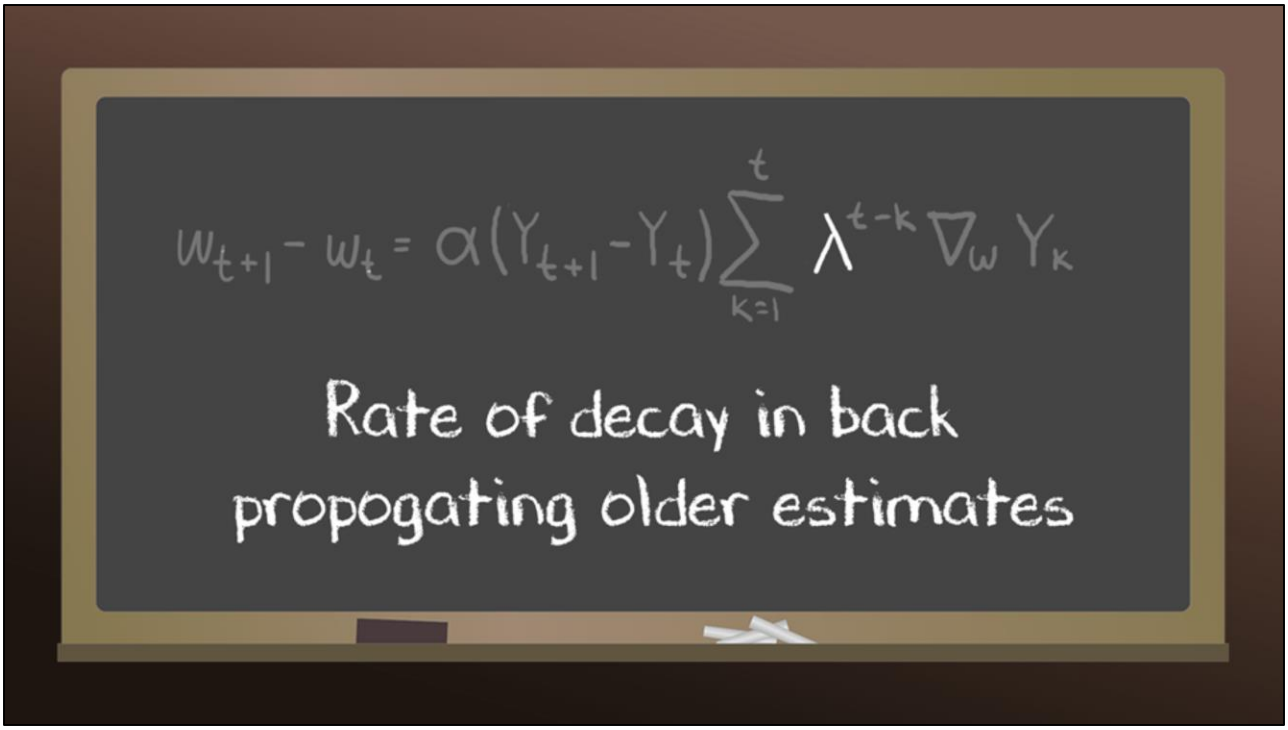
Formula for weight change

This is the formula for changing the weights based on reinforcement learning developed for TD Gammon, and we use something similar in Race. The neural network is setup to update its weights after each turn, to reduce the difference between its evaluation of the previous turn's board position and present turn's board position. This is called Temporal Difference learning. After every turn, the learning algorithm updates each weight in the neural net with this rule.

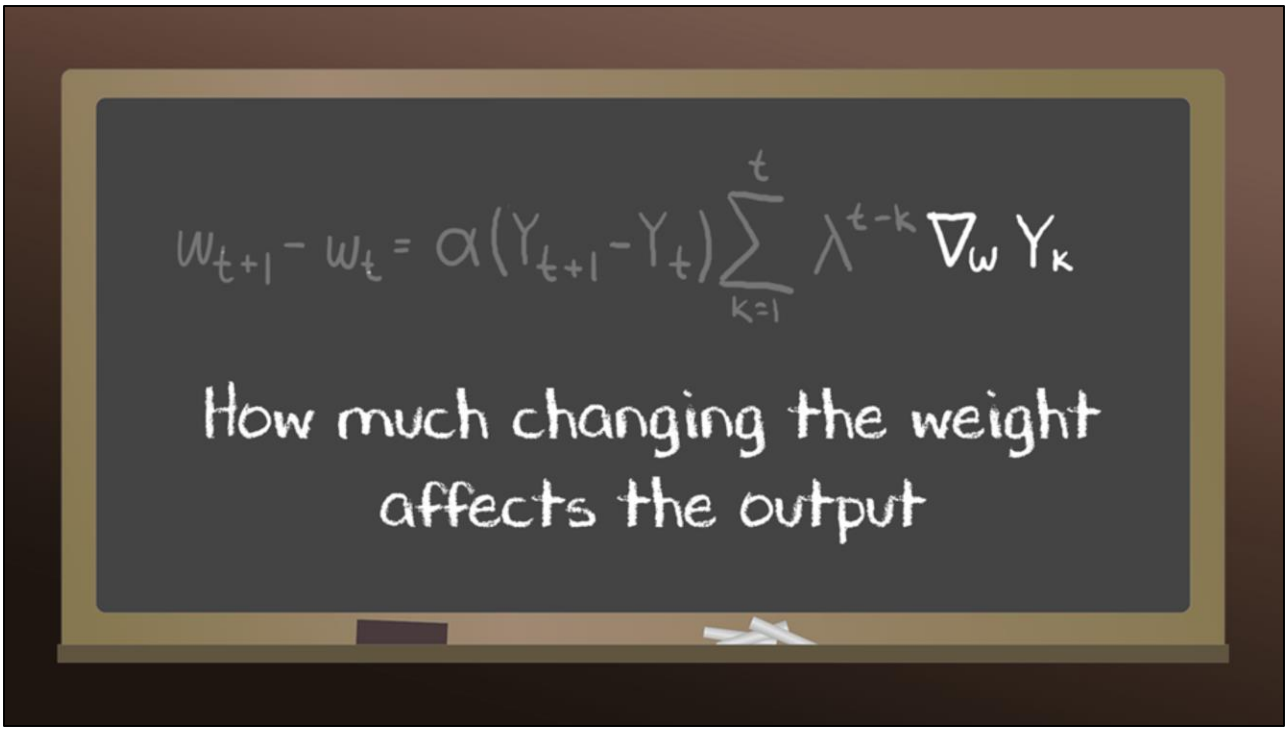

$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$
Amount to change a weight from its value on prev turn

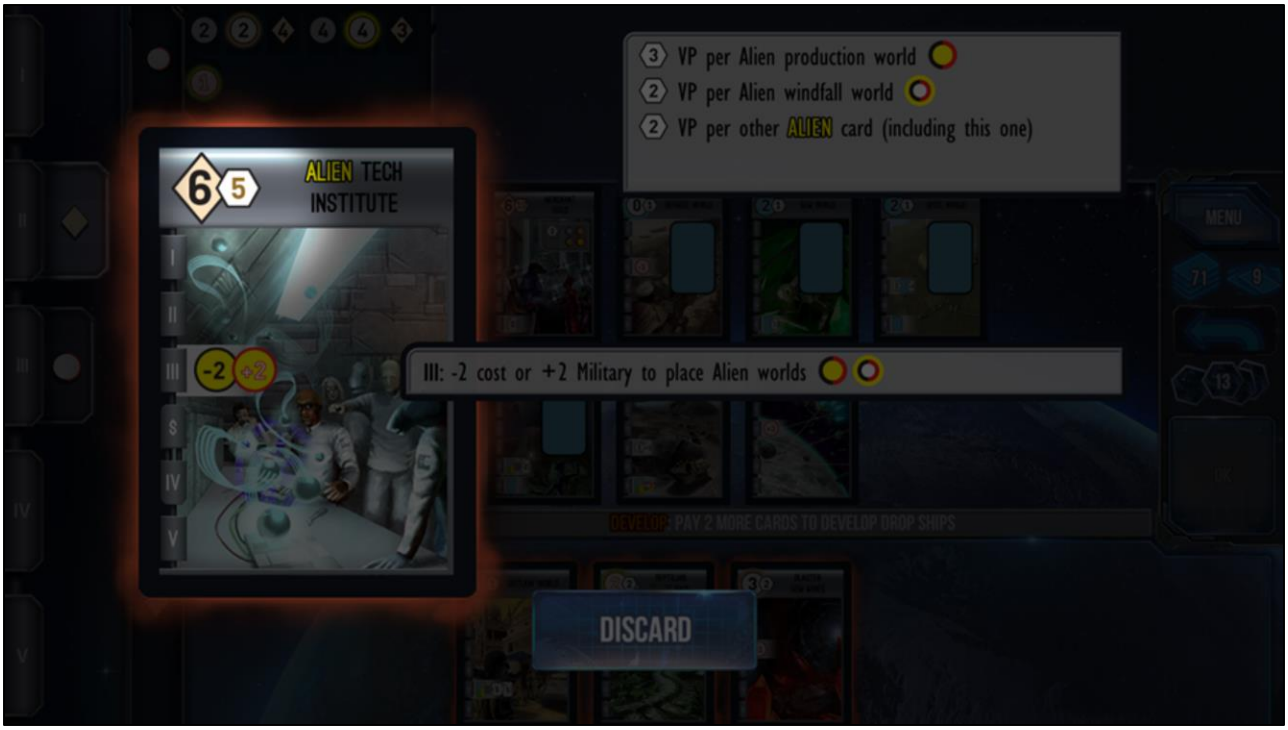

$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$
a learning rate parameter


$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$
Diff between current &
prev turn board evaluations



Lamda rate decaying credit.


$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$
How much changing the weight affects the output

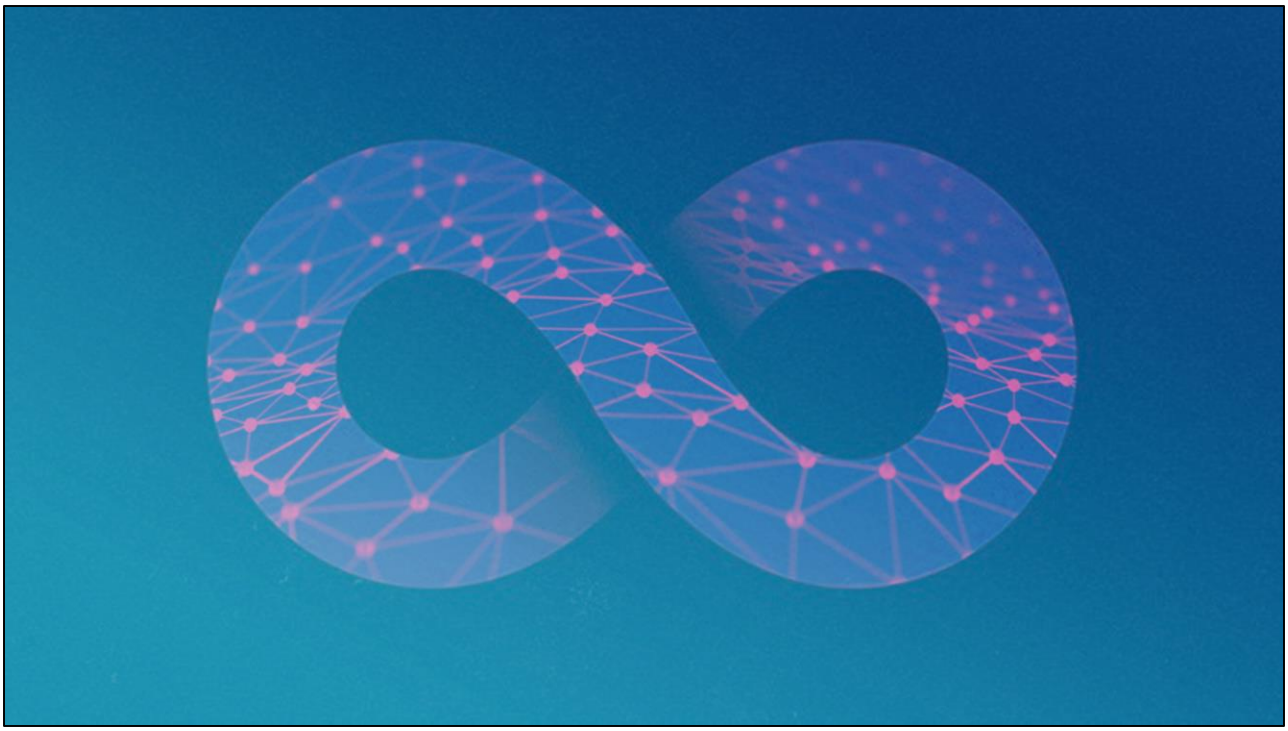


Context sensitive

What's cool about this system is that the outputs are context sensitive. A move isn't always good, or always bad. It's not black and white. The context (all the inputs together) gives enough information to the system for it to determine the output value. So for example, getting a card high in intrinsic VP isn't always the right move. In early game, it might even be worse, because you could be sacrificing an alternative move which might build up your VP engine. In this context, going for VP would be a liability. This results in superior skill over a heuristic driven model which might always favor direct VP gain.

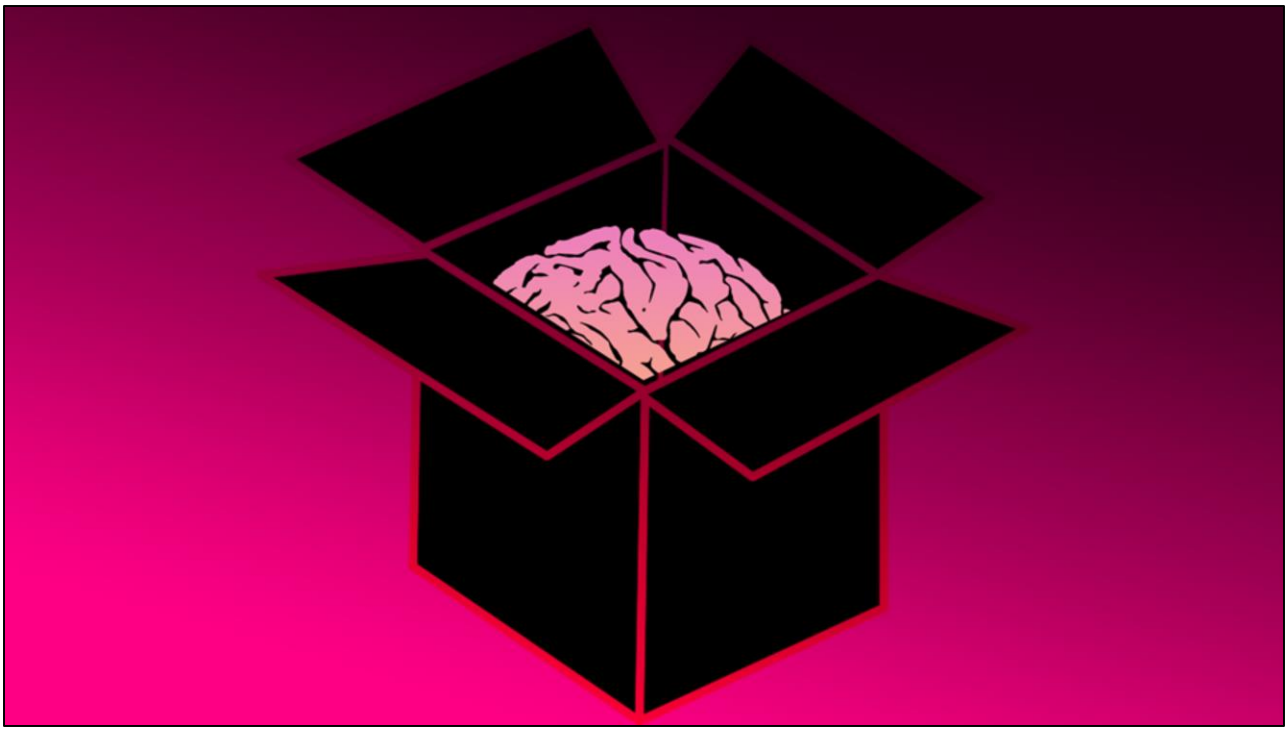
Cats

So neural networks get better by adjusting their edge weights so that the computations result in the inputs and outputs better matching the training data. Training data usually comes from feeding expert data into the neural network. If you want a neural network to be able to identify photos of cats, first you feed it a bunch of photos that a human says are cats. This is expensive. It requires human labor. We do something different, and this is based on that teacherless system pioneered with TD gammon.



Training Data

Where do we get our training data? The neural network actually autogenerates it by playing itself. At the end of the game, it can see if it won or not, and adjust weights accordingly. The calculation is easy, the winner has weight '1' and everyone else '-1.' But it does more than this. interestingly every turn, it just uses it's own prediction as the weights!

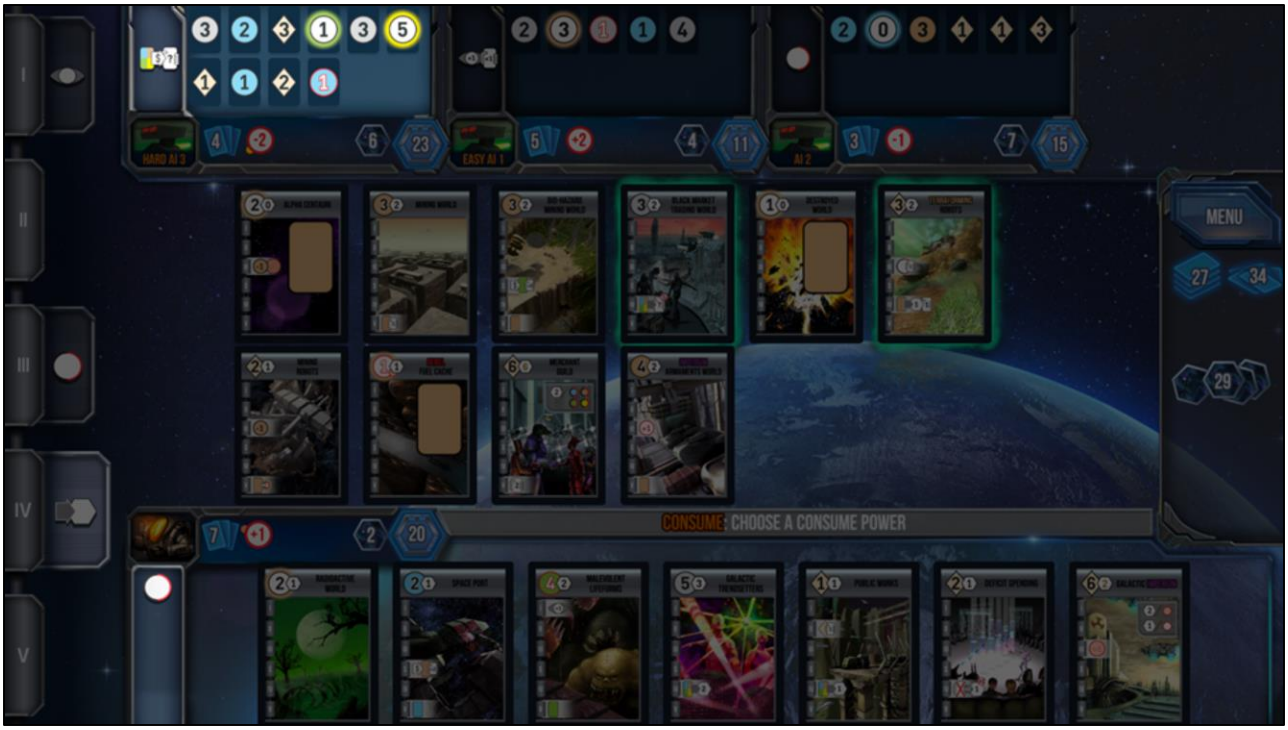


Reinforcement Learning

It’s basically a black box that makes a prediction of what player is going to win through Reinforcement Learning. What I mean by that is, turn n it trains turn n-1. And this back propagates, with that decay I mentioned. So if it's making a prediction of player 3 winning on turn 2, but then on turn 3 it makes a prediction of player 5 winning, it trains itself that on that state on turn two, it should have skewed toward player 5 - it adjusts its weights. At the end of game, rather than using prediction, it does use actual winner as training data, but It's training every time step or turn it gets run. Each opponent effectively pushes the nn. 30,000 games x 4 players x # turns is about a million Time Steps its trained on.

Using this knowledge free system frees us from relying on expert human input. It only needs to know the rules of the game. This is interesting for two reasons. 1) It keeps our costs very low. We can get a ton of training data with virtually no human expense. 2) It frees us from human bias. This has been a big controversy for AI's recently.

As opposed to heuristic driven AI, TD AI are not hampered by what players conventionally accept as correct moves. Because they don't subscribe to the dogma on superior moves; they are able to discover new strategies. As with TD Gammon, we can see emerging strategies the AI has discovered, that then propagate back to the player community. One example is players who play the AI tend to stray from deep rutted strategies that moonshot for particular six cost dev themes.

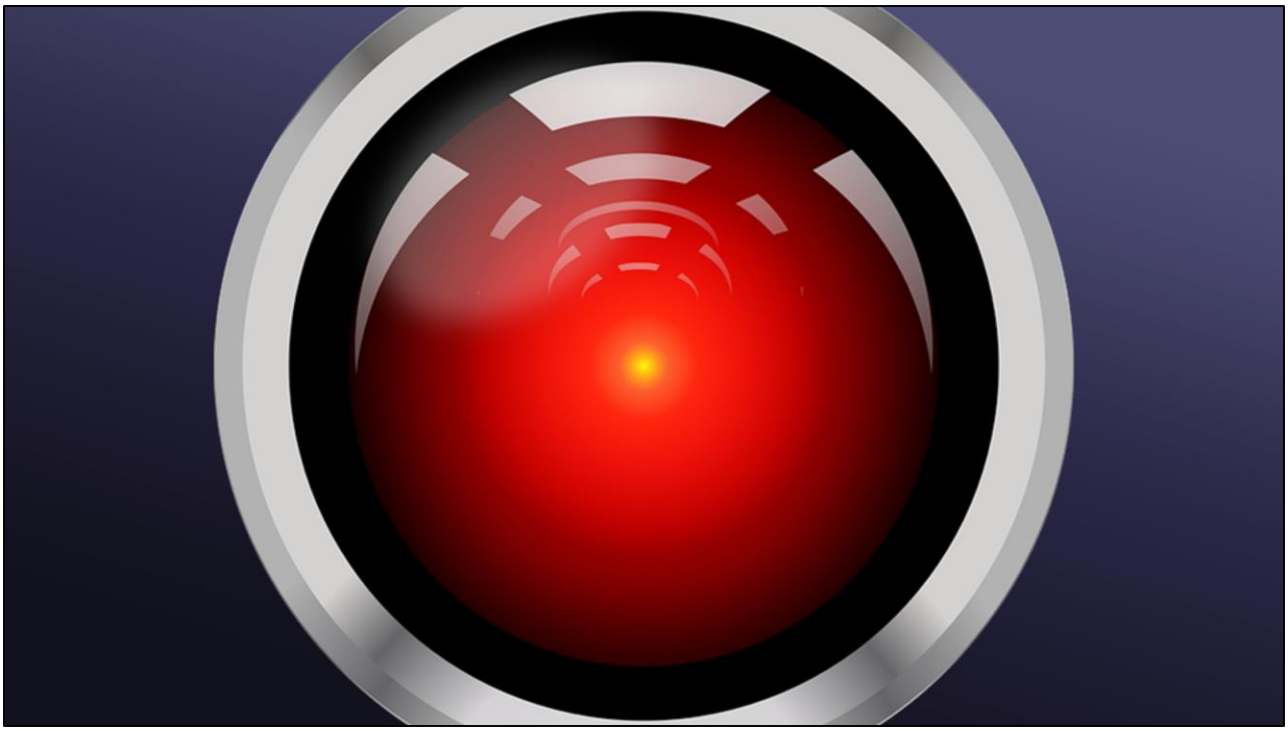


Random to Tactics

What's happening here is that the Initial strategy is random. During first thousand training games, tactics emerge such as Playing it Safe or Piggy Backing. Keldon's AI have developed Piggy Backing strategies, completely independent of human input. So here the AI chose Trade, even though it has no goods to trade. I fell into the trap, chose settle, the AI can settle a world that comes with a good, effectively piggy backing my settle, and trade it.

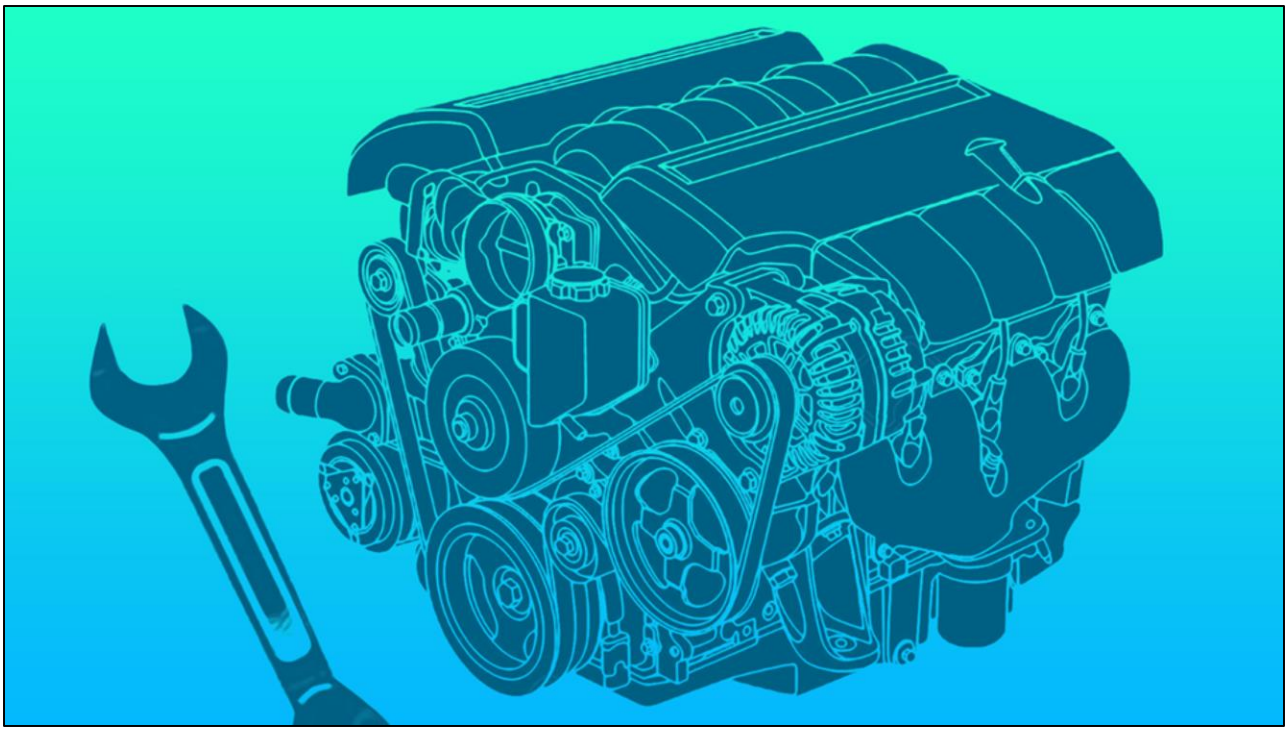
Good Scaling Behavior

Now, the training has already happened. We don't train the AI in the 'live' version because it's already trained on over 30k games, with roughly a million turns, so it's pretty dang good. Training live would slow it down and since it's already 'peaked', it's not clear if training vs the player would improve or nerf it. TD AI has good scaling behavior. At first the AI plays very badly, after a few thousand games, substantial improvements in performance start to crystalize. At some point, continuing to train it will bloat the network without adding significant improvements to player enjoyment.

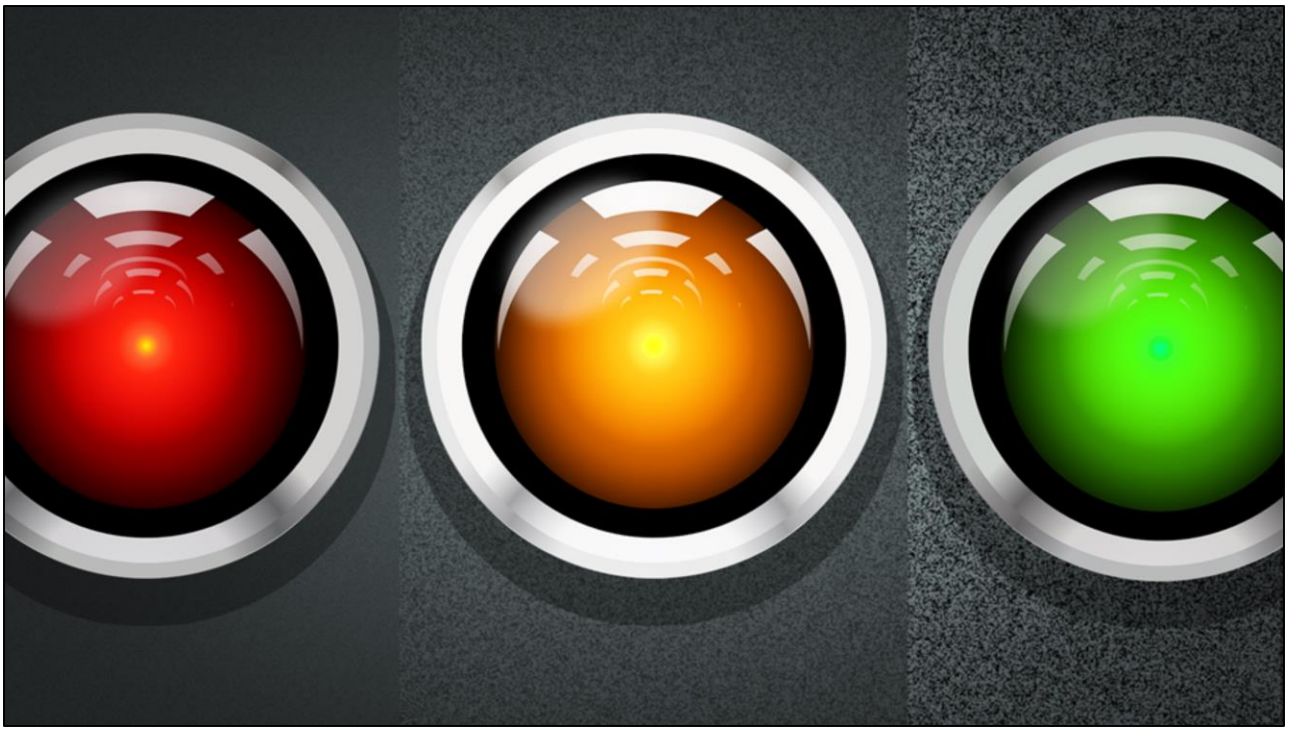


Hard AI

The result of all this is that the AI is really hard. Or it can be. And that's good because it means that your game will be incredibly replayable and provide hundreds of hours of challenge even to advanced players. This hard AI is what makes your game stand up to the heavyweights.



For us, it’s that Hard AI, executing with immediate performance you can feel at 60fps on mobile without tearing through your phone battery, and thats’ UI that communicates the AI actions. We get better performance because we have a custom engine, where we can run our AI on a separate thread, and it works perfectly across platforms.



Difficulty settings

You might think a hard AI is not ideal for some people, maybe new players. And you're right! It's very difficult to make an AI harder, but you can nerf an AI to be easier pretty simply. In our neural network outputs, we add noise to the score. Increasing the noise will make the AI choose the 2nd or 3rd best options. You can tune the amount of noise you add to find the right difficulty settings. When determining the noise for a medium AI, you want it to win roughly 25% of games against a hard AI.

AI UI Timing

When you incorporate an AI into your game, it may be taking turns at lightning speed. To be competitive, you need to slow down, and let those moments sink in for the player. You can use UI to emphasize and de-emphasize things that are happening in the game. If we paused by the onesies to show everything that happens you wouldn't tolerate it. Some games do this. They show *everything* and maybe have an option to make it go faster. That has advantages for clarity and disadvantages for monotony. We started this way too, we thought of everything as a script that executes in order, but it got unwieldy. So we asked ourselves to dig deeper on what we are trying to do with this UI. And our answer was we are trying to communicate, with visuals and effects, the state changes of the game. And what's the major bandwidth constraint in that system? Player attention. What should they pay attention to?

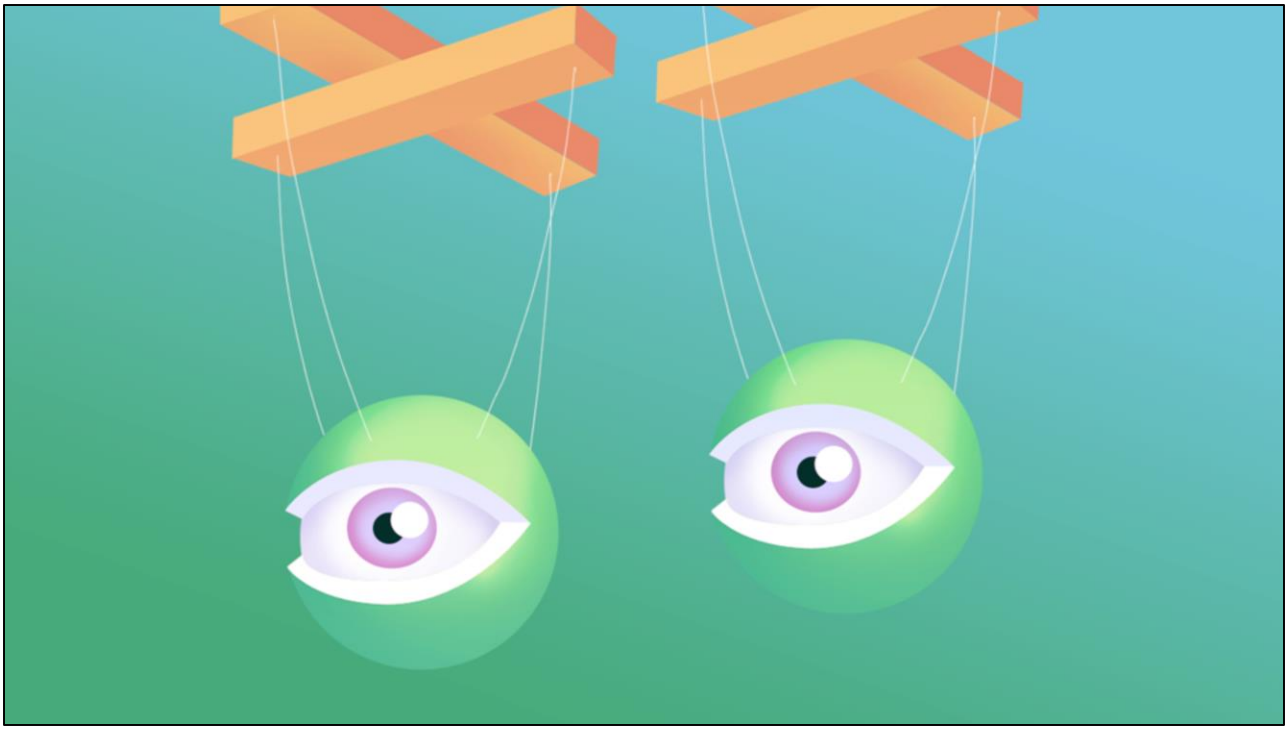
Conducting events

In race, the turns happen simultaneously. Everyone reveals and effectively executes their turns at the same time. We could show each action as a vignette, but that breaks the spirit of the simultaneous play. As a UI designer, it feels like you're a traffic conductor on a complex intersection where some things intersect, and others don't, and some things stop, and others go, and others back up! Managing the timing so players don't feel overwhelmed or bored is a careful balancing act. Each animation isn't necessarily competing for attention, it's competing for cadence.

Locking tokens

We have a locking attention tokens. If we have an action that's important that the player notices, for example an AI settles a new world, then that action grabs the Main Attention token. While that action is playing out, other 'important' actions are stalled until the main attention token is released. after the animation plays, the token is released, and the next waiting action grabs the token and proceeds. In all there are about 10 locking tokens, for things like the draw deck, vp pool, discard pile, as well as main focus area.

We have fine grain control of the clocks and timing because of that custom engine. It doesn't waste time / power.



Flow

Communicating your AI's actions through a system like this will help manage the flow of the UI animations on the screen. UI isn't just about where you place things on the screen it's the cadence. Your users won't know why your game feels good, they just know that they want to play it again, which is how you get to someone playing 6000 games in 2 months.

So, I've talked a lot about the AI's we've made, how they apply to boardgames, how to keep them fast, and how to communicate their moves. I want to talk to you a little about how to choose right AI mode for your game. You can use AI to extend the depth and scope of your game to compete with larger titles. So, how do you know if you can you use a temporal difference AI? This kind of AI will greatly improve replayability of your game, but it won't work for a lot of different titles. I can give you a checklist to measure whether your game would be a good candidate!

Boardgame

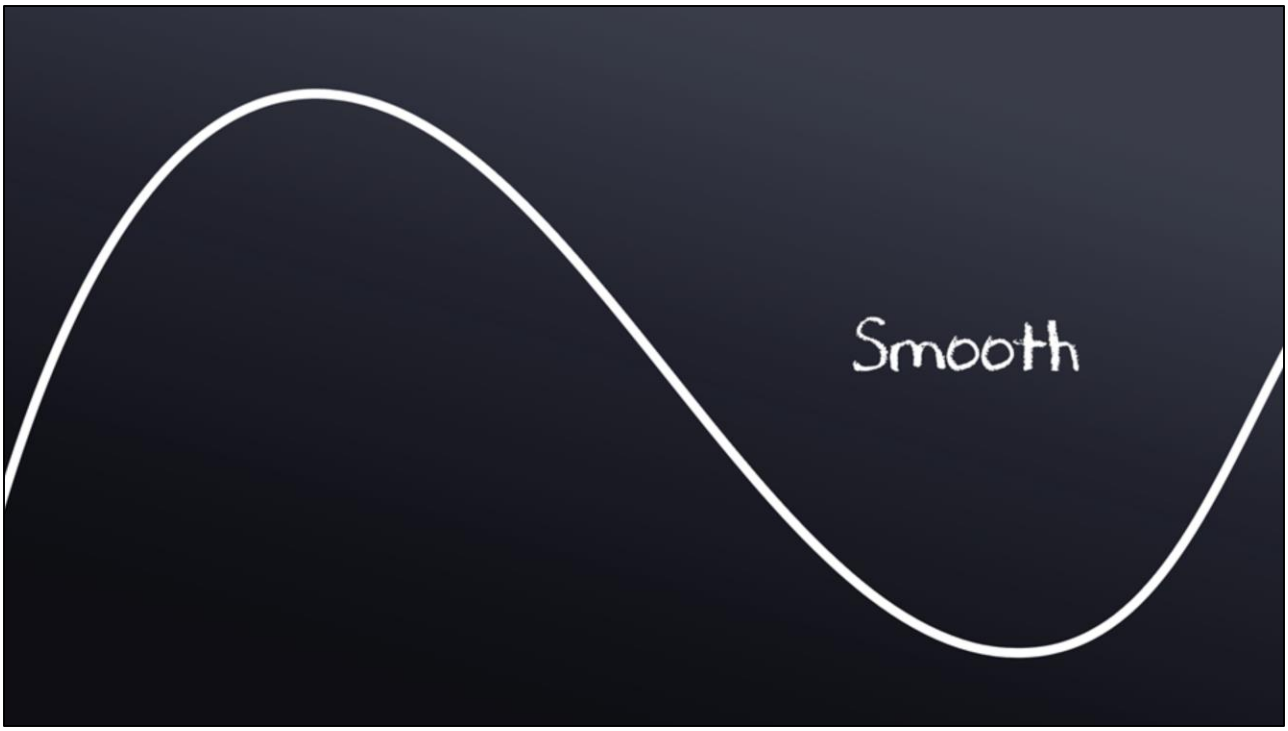
First off, is it a board game? It doesn't have to be, but boardgames work really well. Why? They require complex sophistication to play at expert level but rules are well defined. Discrete rules are critical to make a neural net game AI.

Stochastic vs Deterministic

Stochastic games, games with high RNG are good candidates. That means dice rolling, deck shuffling. The randomness means there's a lot more probability space to explore, which means more chances to discover novel strategies. That makes a game like Race, with a deck of cards, a good candidate for a TD AI. Deterministic games, like chess, with no randomness would be bad, because there is no noise generated by stochastic RNG, resulting in a very limited probability space to explore. If you wanted to use a TD AI to learn a deterministic game theoretically you could apply external noise to get the AI to explore new choices.

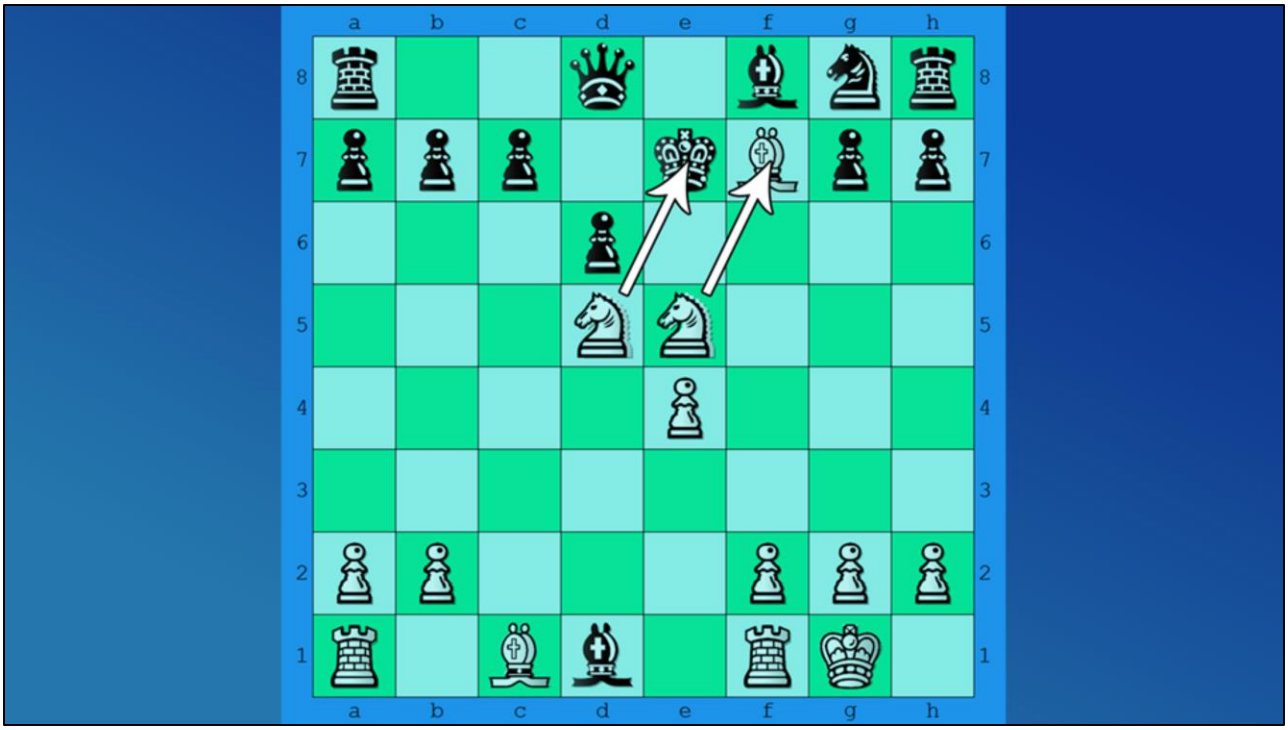
Bruteforce Branching

Furthermore, because games like Race have a lot of randomness, there is a high branching ratio of possible outcomes for RNG variables, which means a deep search method (as traditionally used with deterministic games like Chess and Checkers) would not be feasible even with a supercomputer to cover simulating through the space.



Smooth

Using the TD AI on nondeterministic games results in a smoother continuous function. So your neural network is acting as a function. And small changes to the input state in a stochastic game will result in small changes to the probability of winning. This makes the game easier to learn.



Bad for spatial

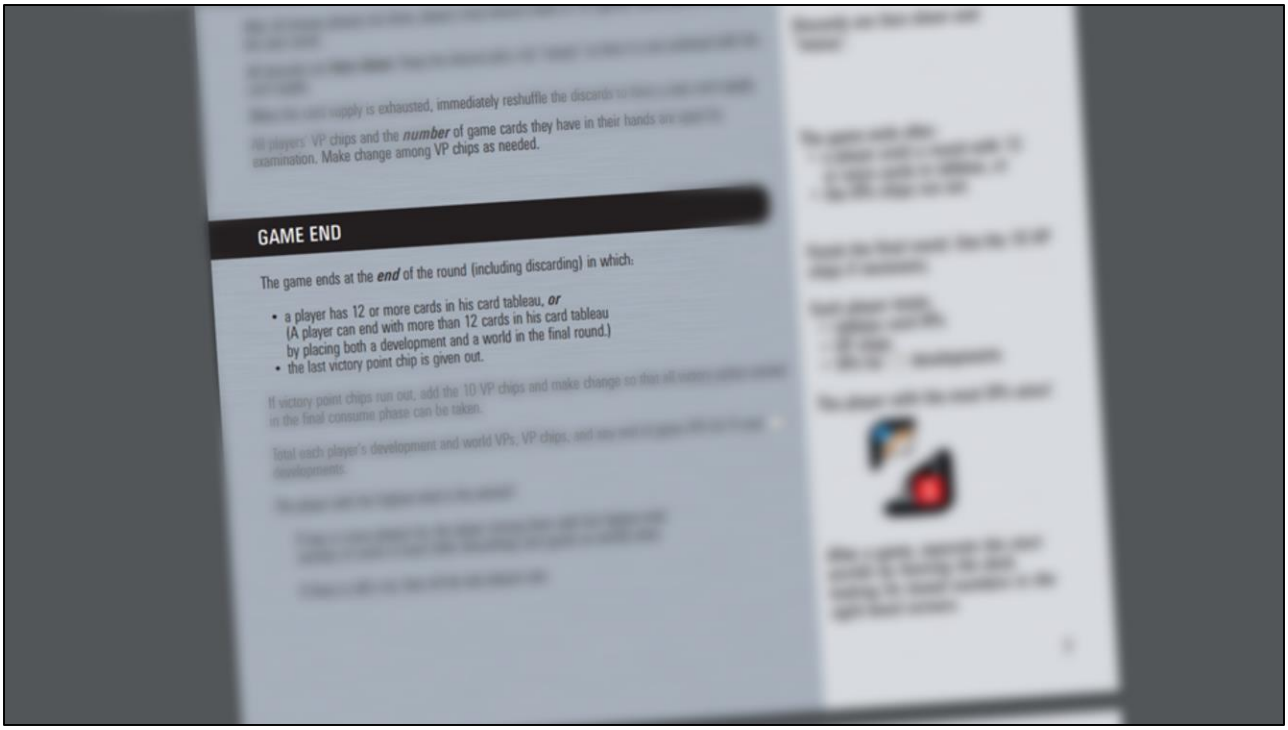
The TD AI is better on games where board position of piece relative to another is less important. So Othello, Go, Chess are not good candidates because the relative position of pieces (especially in canonical patterns) matters more than discrete input variables. This is the Sea Cadet Mate, which achieves mate in 10 moves. Identifying spatial arrangements is not this AI's strong suit because it's expecting a small variable change (such as one piece being moved) to correspond to a small victory probability change, which is not the case here at all.

Judgment not Calculation

TD AI seems to breakdown when a high degree of precision is required, for example in chess to reach checkmate at the end of the game often requires a very specific set of moves that are 'obvious' to human players but would be difficult for this type of AI. Interestingly, games with strong strategy or judgment but low meticulousness or calculation, are the ones this style of AI is good at.

Terminal States

A game that pushes toward a definitive ending or a terminal state is a good candidate. In Backgammon, the checkers are on a one way track, the game will end. In chess though, the game can become stalemated, with pieces repeating a dance around each other. A neural network could become stuck in a strategy cycle that could last forever. In this case the network couldn't learn as it wouldn't receive the final reward signal.



Rules Flexibility

You could get around this. In Race, this infinite turn issue was the case, and Keldon included a new game end rule for training, that the game end in 30 turns. You may or may not have the flexibility on your game design to suit your game to a TD AI.

Multiple Turns

Although you could theoretically make a TD AI for a coop game, which would be an interesting challenge to try to create something helpful, that doesn't help too much, it wouldn't work well for a game like One Night Werewolf. The reason? Delayed rewards are the crux of TD AI learning. The game needs to last for multiple turns in order for the temporal difference algorithm to back propagate rewards. In fact, as the game approaches the end, the AI acumen falls off because it can't look ahead. There are a lot of endgame errors. These errors can be reduced by adding some heuristic driven decisions toward the end of the game, but that has a human cost.

Simulate Environment

The AI will need to simulate the entire environment. This means there needs to be a fixed number of inputs with discrete legal moves. With a fully simulatable game you can generate limitless training data. Which means you can make it really good. Cheap and fast. A game like Tag wouldn't work because the turns are not discrete. The environment is too complex. Too many unfixed variables, like how many players.

Existing base

Finally, a good candidate for a game with a temporal difference AI is a game with an existing player base. How can you find expert players to test your neural network? Find the player community. This way you can evaluate your AI not just based on them beating other AI, but also by beating benchmark expert players. Games that already exist in one medium, such as cardboard, but not in another, like digital, work well for this. A board game that hasn't yet launched may be a poor candidate, because there won't yet be a pool of expert players. We launched the Race beta on BoardGameGeek.com because it hosts a lot of expert players. The downside to relying on expert players to evaluate your AI is that benchmark humans are super slow compared the benchmark AI players. But it's still helpful.

Bug hunting

Players can help you identify a problem. Most likely it'll be an error in the simulation. Most bugs with this kind of AI will result from flaws in the simulation that the AI runs through not matching what happens. The main way to fix this is just tediously bug hunting prediction code to make sure it's not making boneheaded errors.

IANAAIR

What's in the future with this? Well, when speculating, I am not an AI researcher.

Race in VR.
For me, the next challenge is we're going to bring Race for the Galaxy to VR, which means we'll be tasked with bringing this AI to life for players. The AI won't simply execute, it will animate, emote, and give visual feedback on those high level state changes.

In the future we could use neural nets to generate AI audio and facial animations.

Roll AI

As a small company, you can compete by driving hard in a niche. We're going to make the sister game, Roll for the Galaxy. The challenges The challenge is that on a turns are several more decisions deep for each player (or AI) to make. Rather than just choosing a roll, the player is assigning dice to slots, and many of those dice have multiple optional functions. The issue here is that the decision tree gets very deep, which will create performance issues when the AI simulates through all possible turn choices. So, stay tuned for how we figure that out.

Mainstream

Twenty years ago, it wasn't necessarily clear what the practical applications for temporal difference AI was, but as board games get more popular, and even mainstream, this kind of AI is going to take off. It's worth figuring out. It's cheap. And I'd love to play more games that use it!


There are No Weight Classes in Game Dev:
Competing Alongside Goliaths
Game AI North
@tduringer
TEMPLE
GATES