
DARK SOULS III
GG Too Easy
OPLOLREPLAY

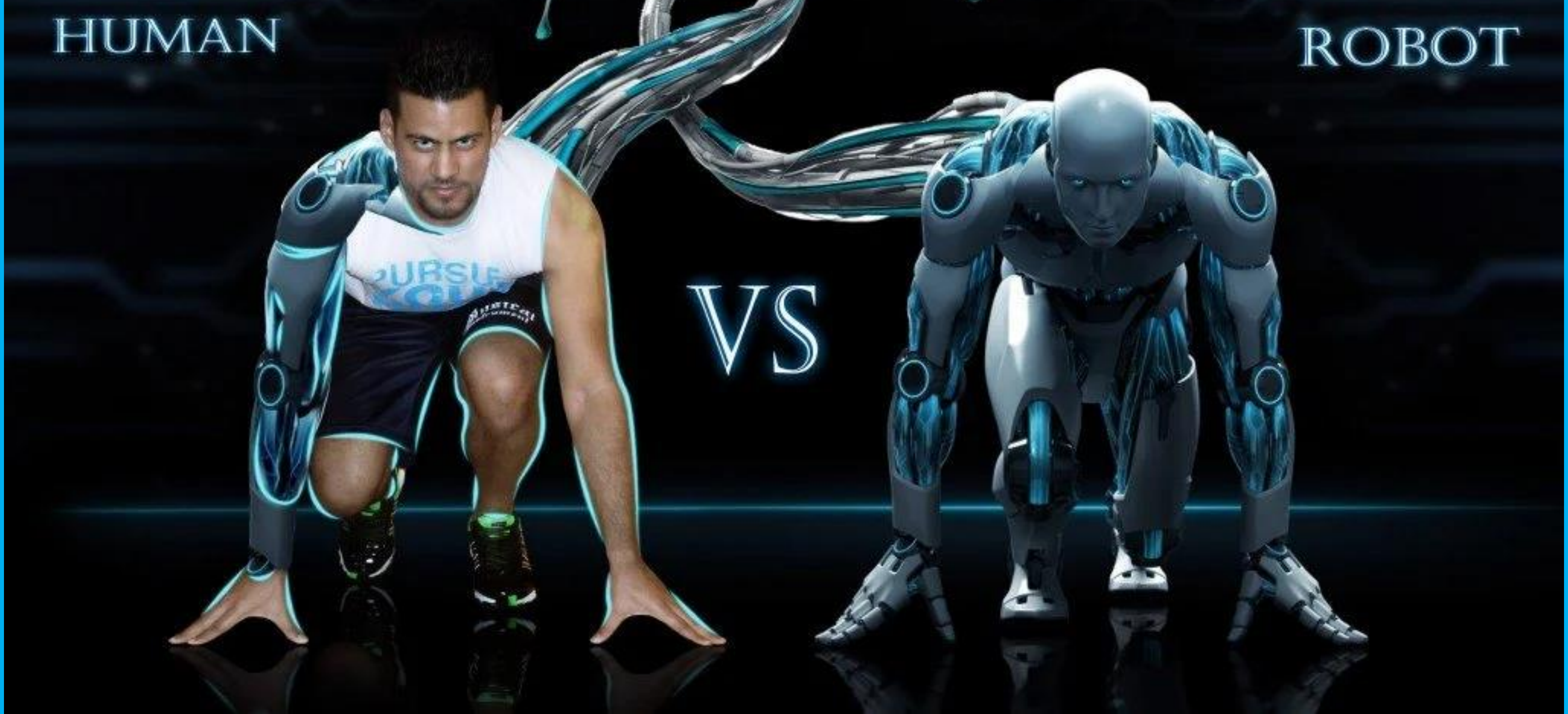

Quotes About Not
Having Friends


FORTNITE

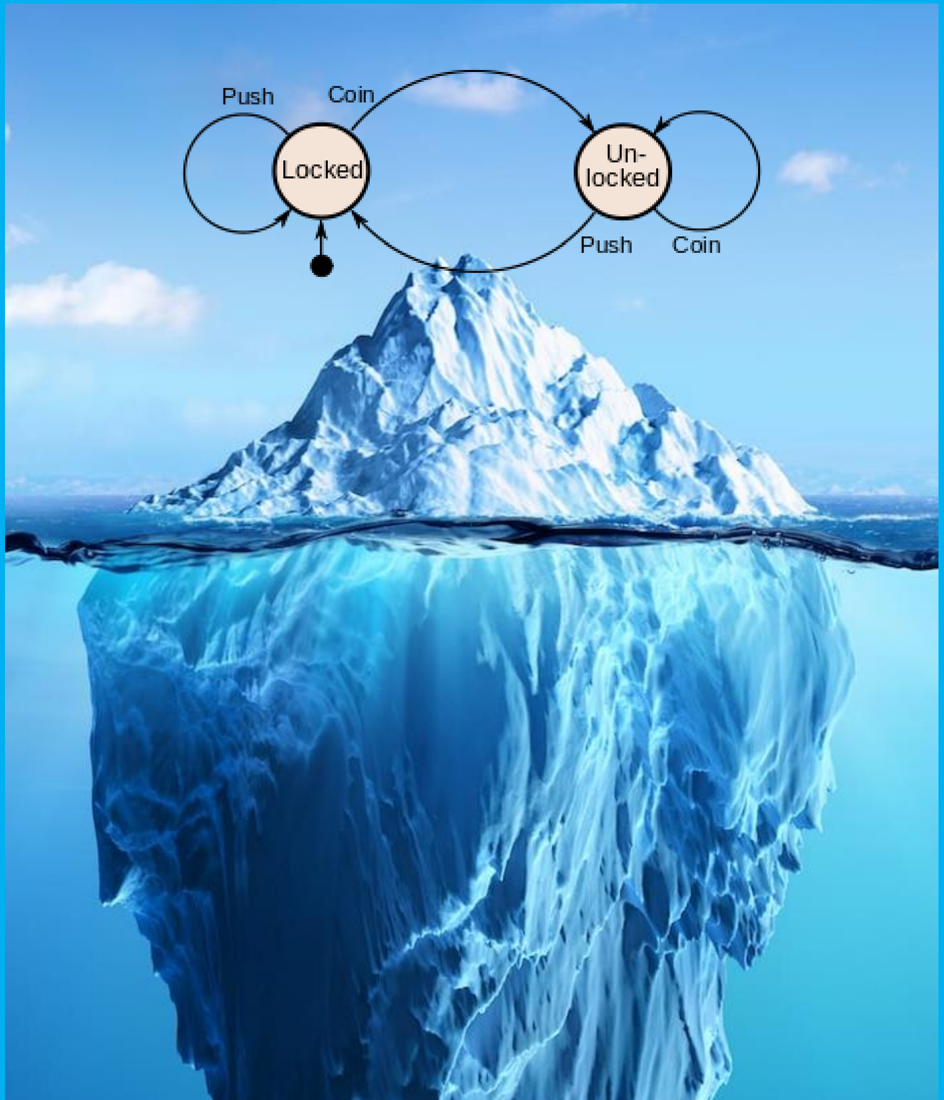

WE WANT YOU!


Make an AI!

How to make an AI
that will whoop your butt in Video Games?

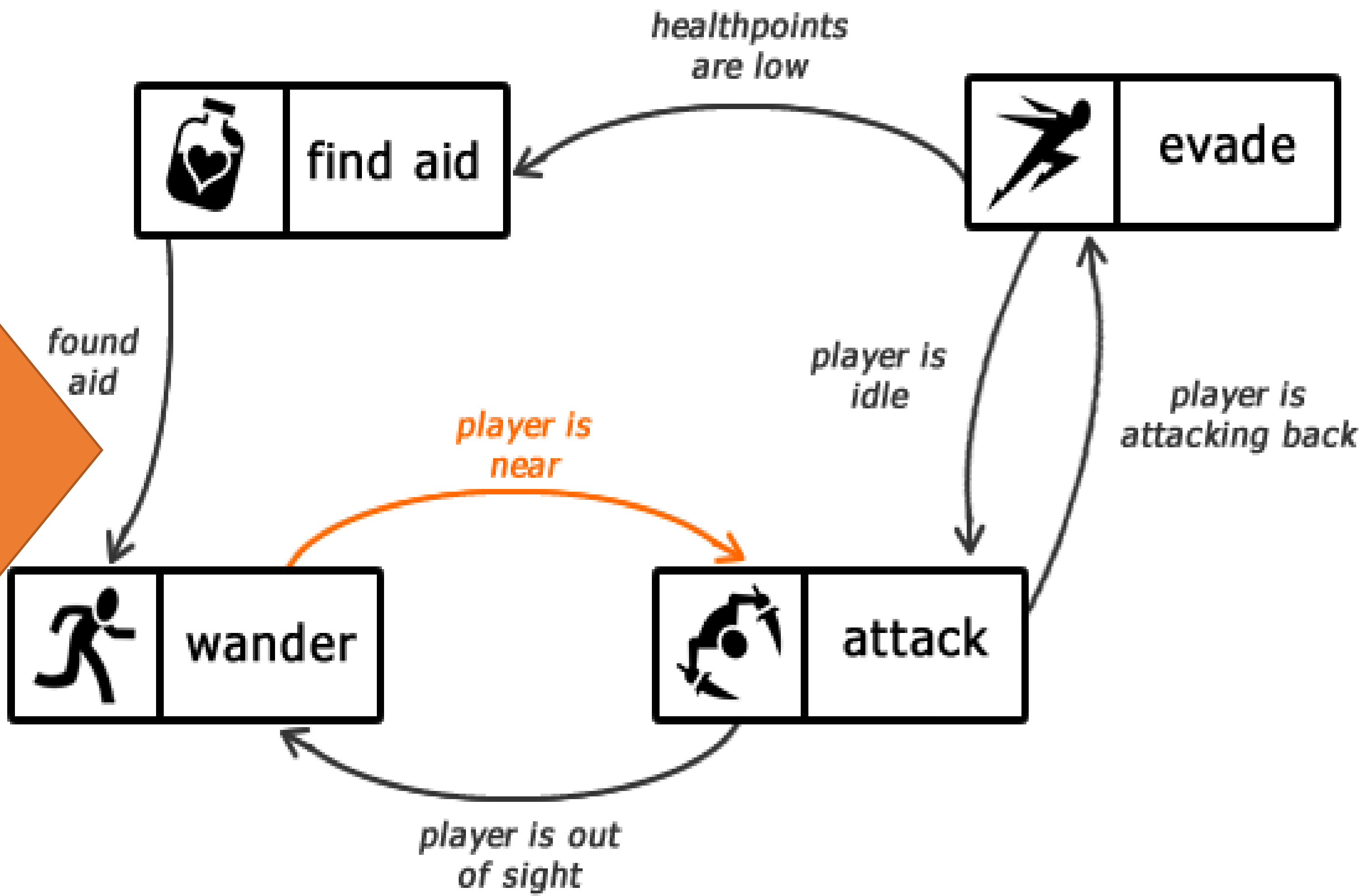


Video Game AI Iceberg



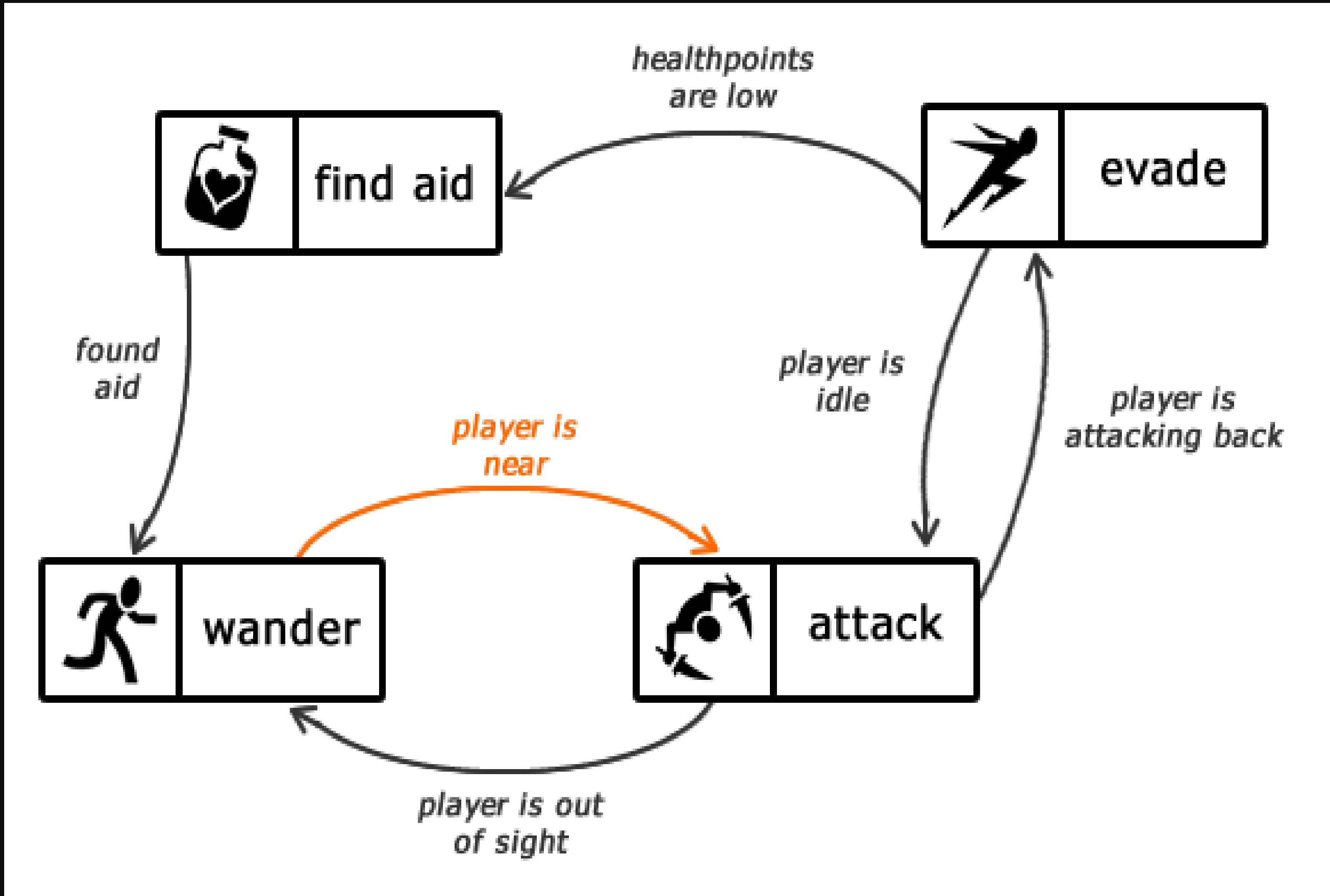
Finite State Machines

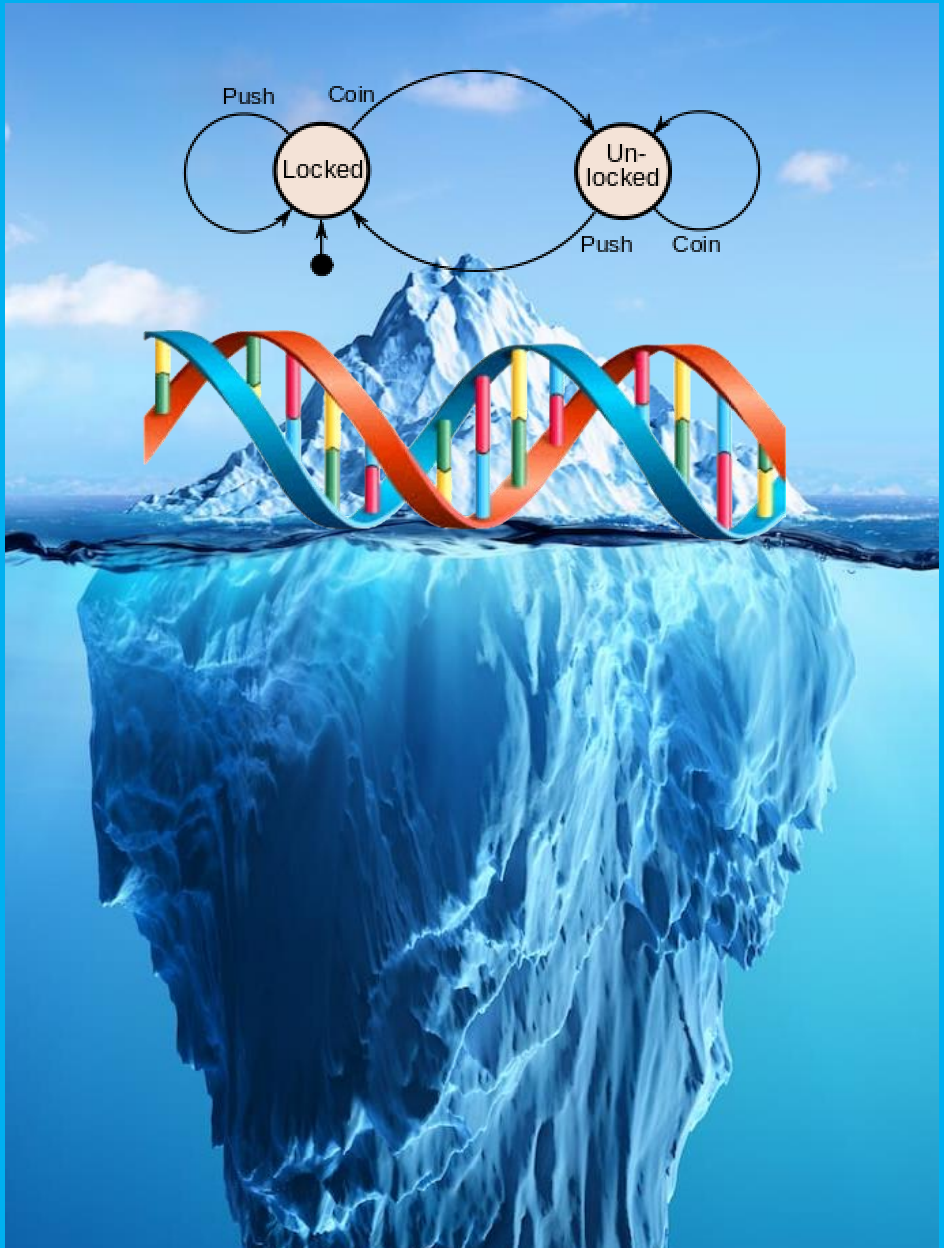
How to design?



Benefits:
- Easier to adjust difficulty
- Understandable
- Easy to balance
- Simple to simulate
- Can be created in the toilet
- Cheap to develop

Drawbacks:
- Long development time
- Players don’t like it
- You don’t like it
- Boring
- No „logic” – just pure code
- Non – adaptable
- Doesn’t look cool on your resume

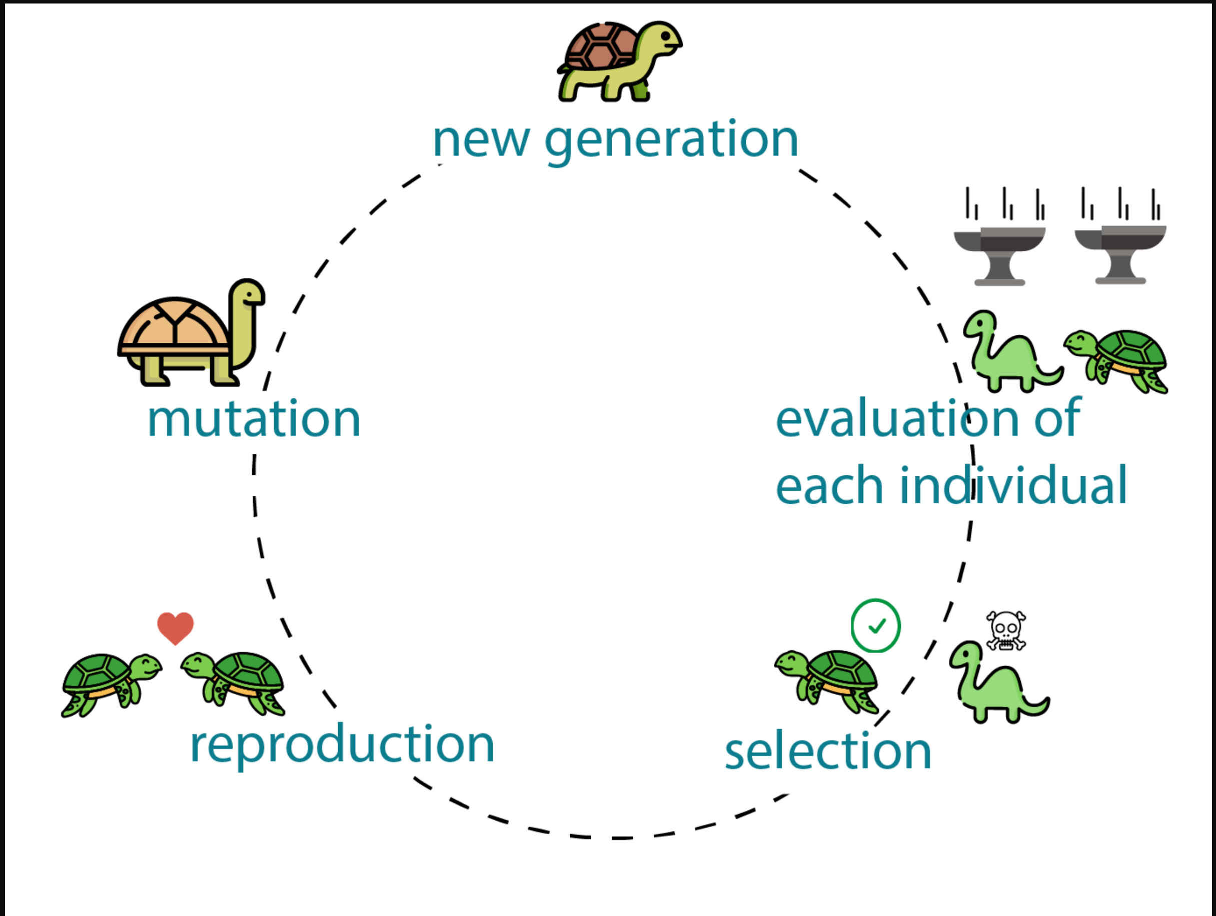




Genetic Algorithm

How to design?

- Whenever a specific event occurs (Text input, x frames passed), AI performs the next action from the list of actions it selected
- Simulate a lot of randomly generated AIs, and measure their performance
- Select the top k best performers, and mutate/replicate/breed them until a satisfactory score is obtained
- In the simplest form, there is no need to understand what is happening by the AI (no input required)

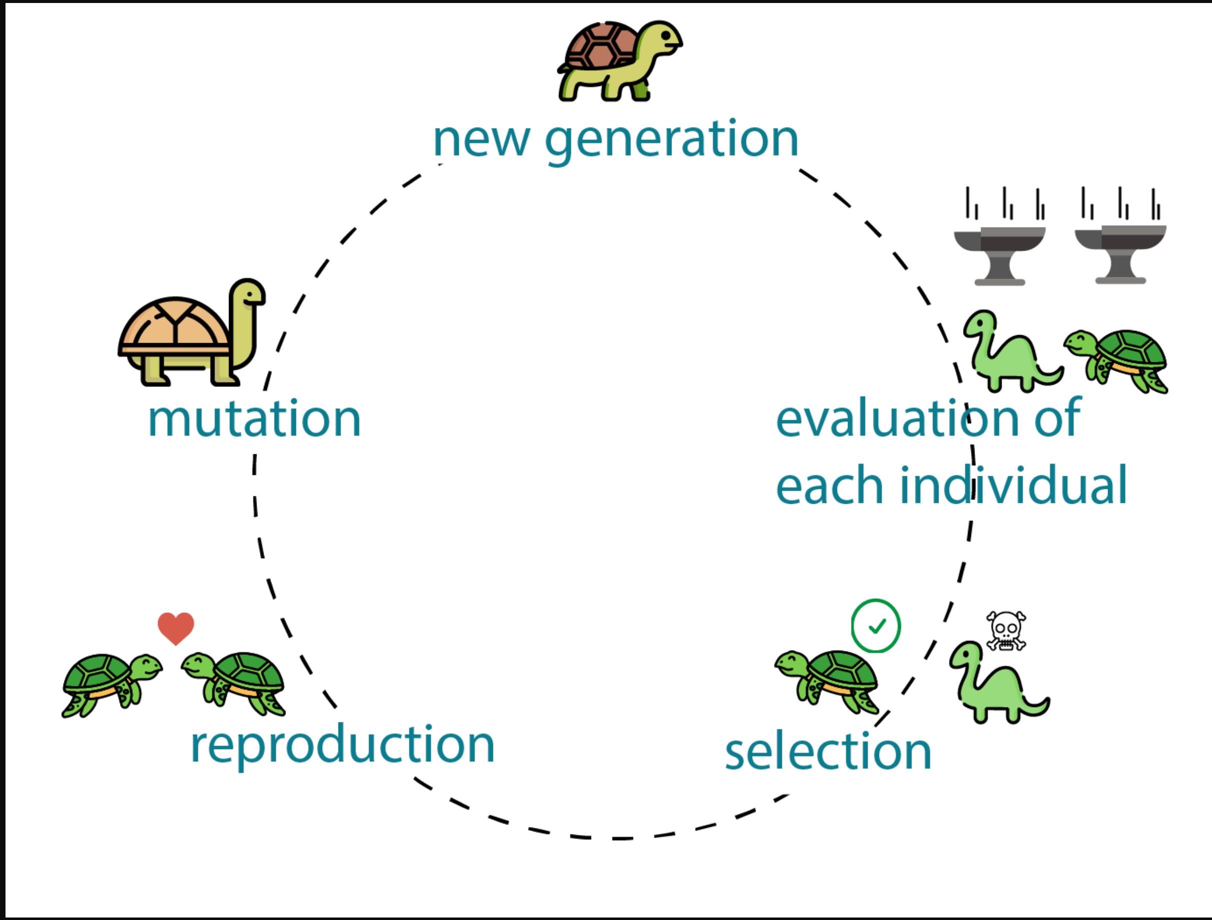


Benefits:

- No input required
- If there are no random variables, this solution can work
- Easy to code for a YouTube video

Drawbacks:

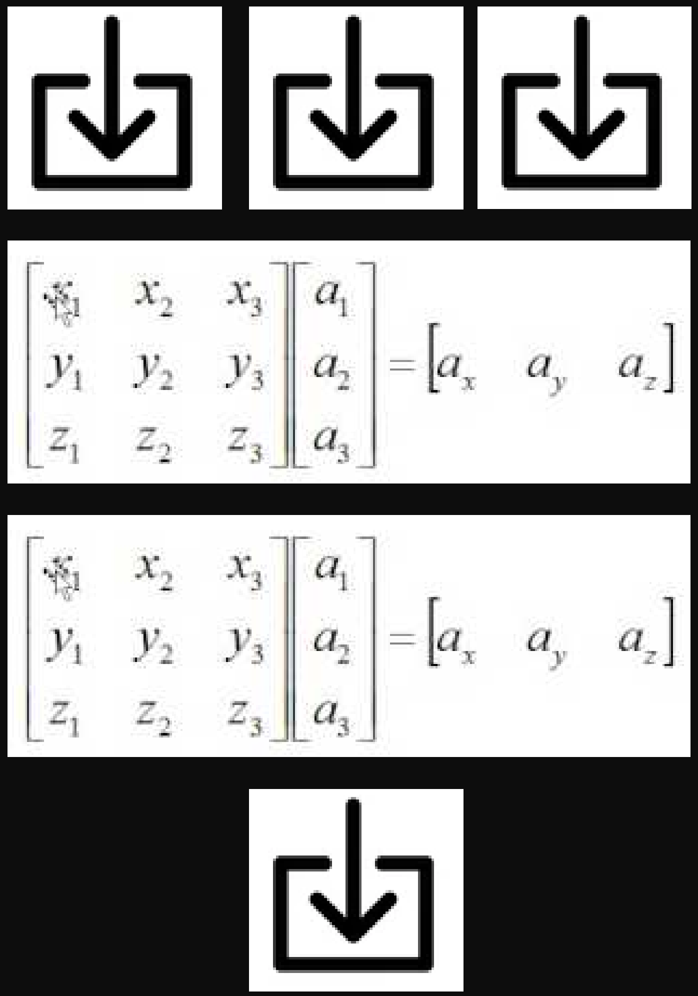
- Pretty much as stupid as it gets
- Does not adapt to the situation



Genetic Algorithm…
but on steroids
(NN based)

How to design?

- Initialize a small Neural Network with random weights, generate a population
- Whenever a specific event occurs, feed each AI current state (inputs)
- Evaluate the performance (survival time, response quality, ect)
- Mutate best performers
- Stop when satisfactory

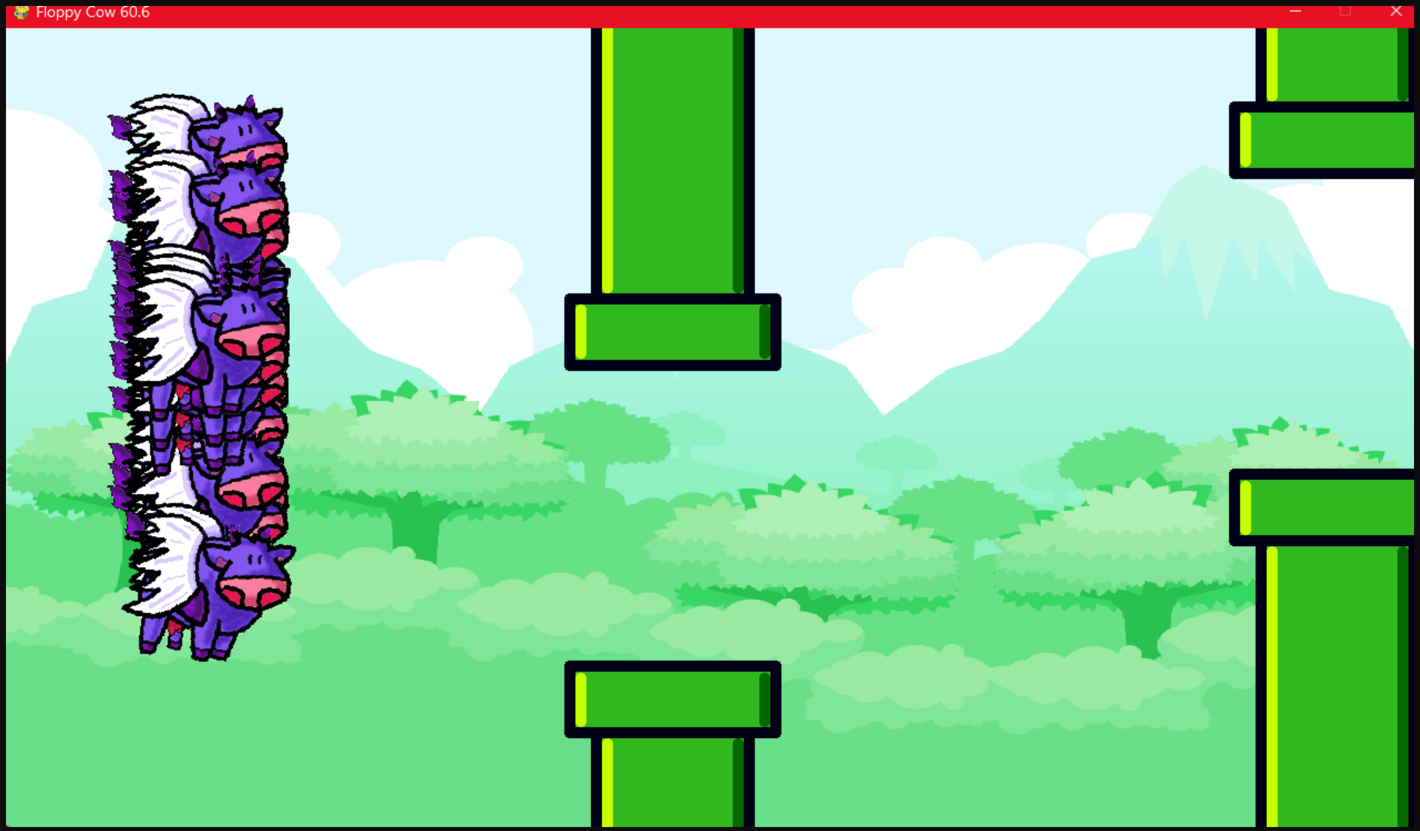


Benefits:

- Reltively easy to code
- Can learn (kind of) advanced concepts
- Can adapt to the situation
- Pretty much all YouTube videos use this algorithm
- No need for complex loss functions

Drawbacks:

- May require many re-launches to get the network shape right
- Requires access to the game source code to amplify training time



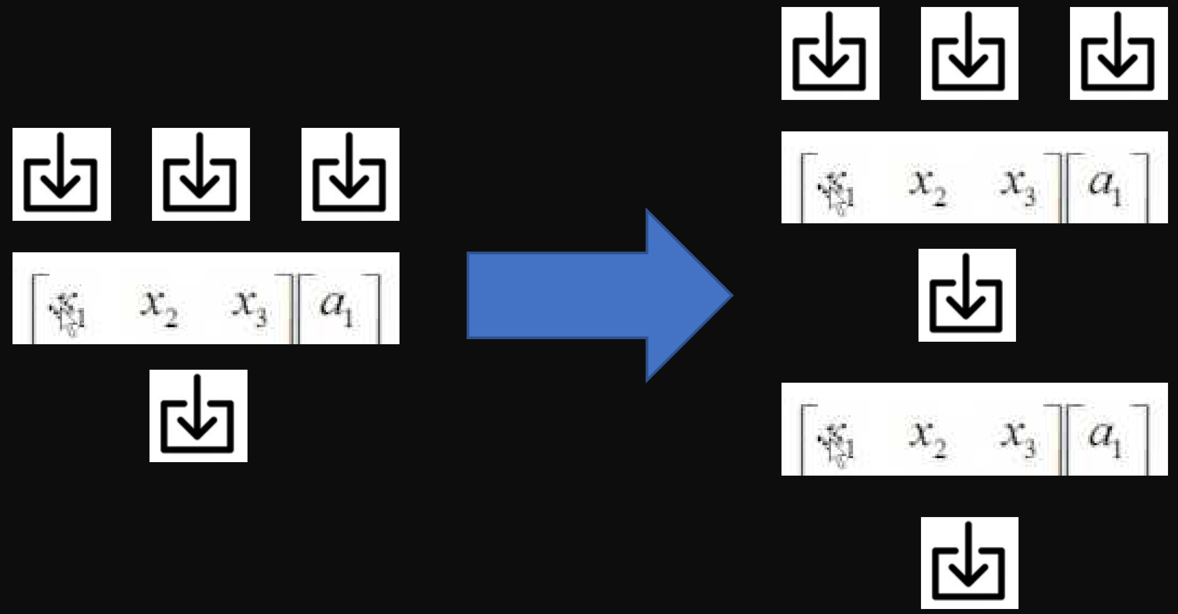
A pretty NEAT algorithm

(NeuroEvolution of Augmented Topologies)

Paper: https://nn.cs.utexas.edu/downloads/papers/stanley.cec02.pdf

How to design?

- Initialize a small Neural Network with a single neuron
- Whenever a specific event occurs, feed each AI current state (inputs)
- Evaluate the performance (survival time, response quality, ect)
- Mutate best performers in various ways (list of mutations in next 2 slides)

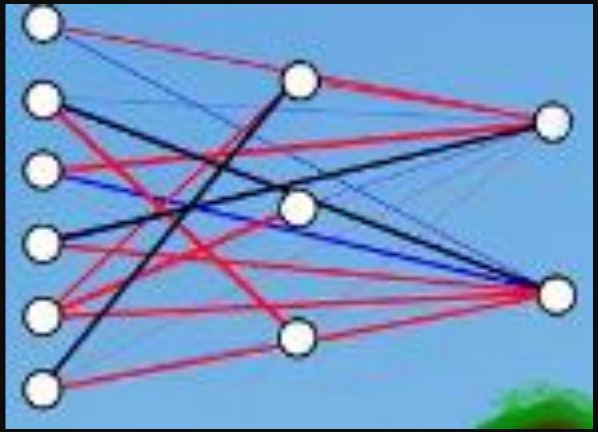


NEAT bonuses

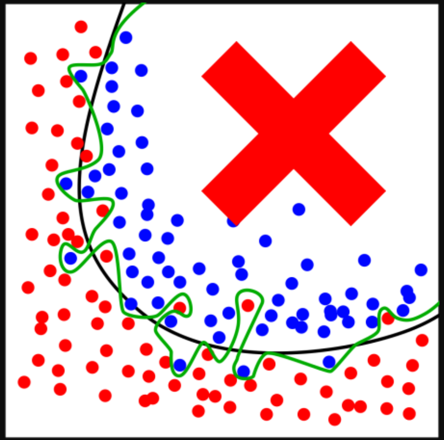
NEAT library can do most of the job for you

The most overused algorithm for simple games

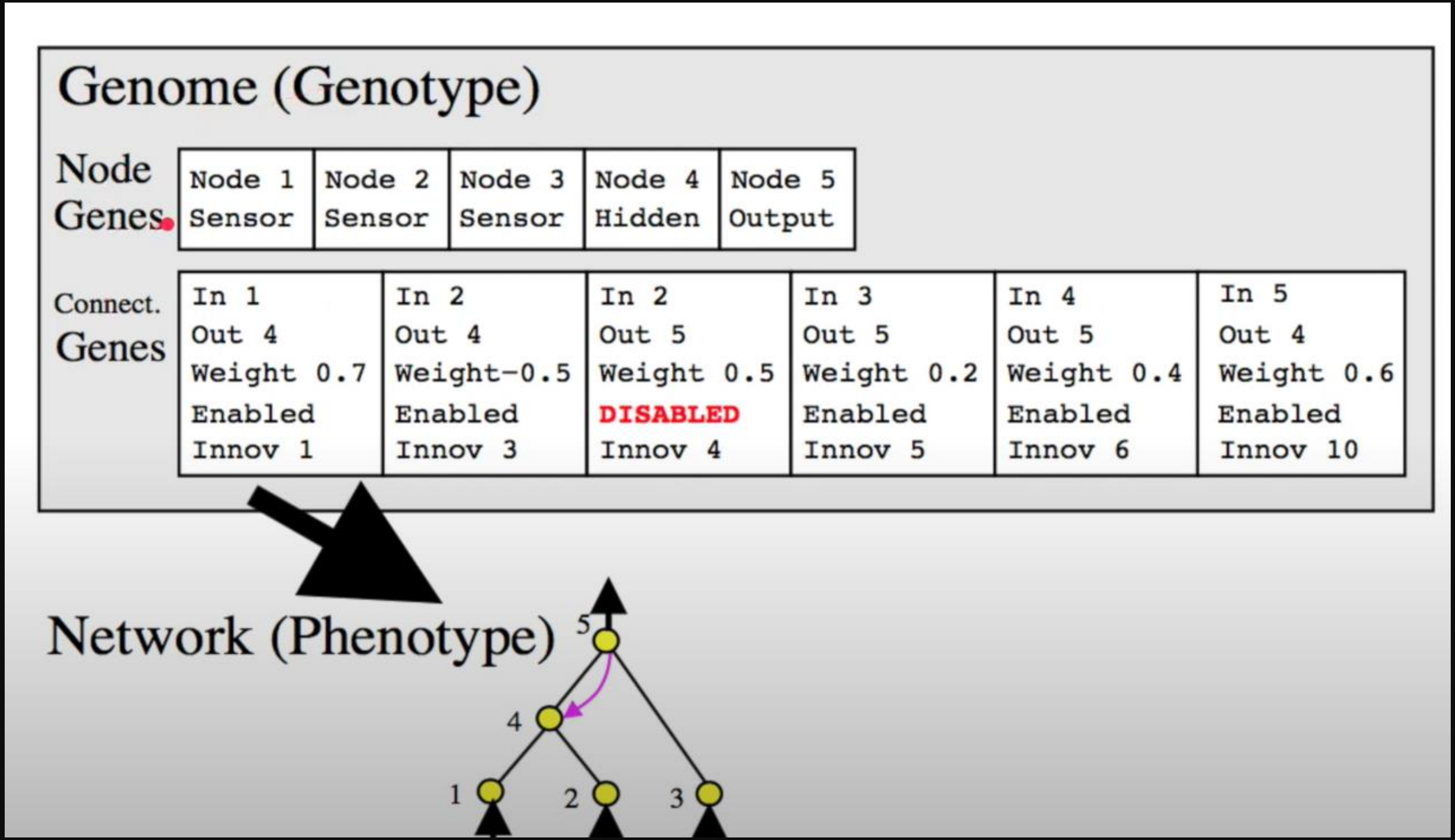
Visualizations are awesome



Very hard to overtrain

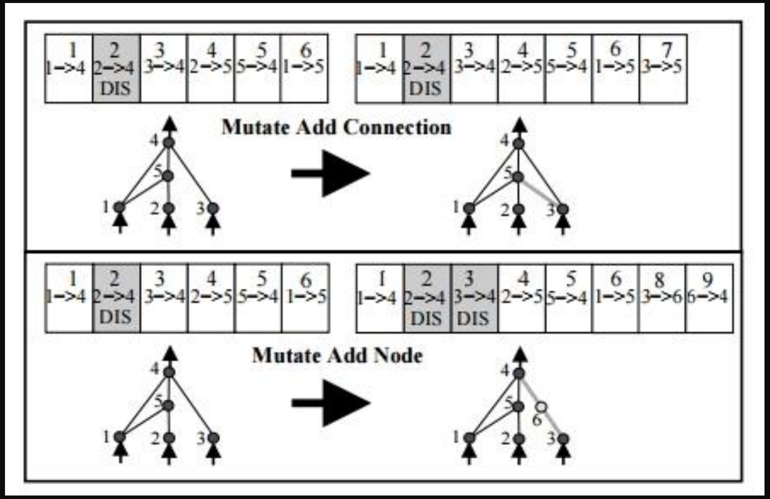


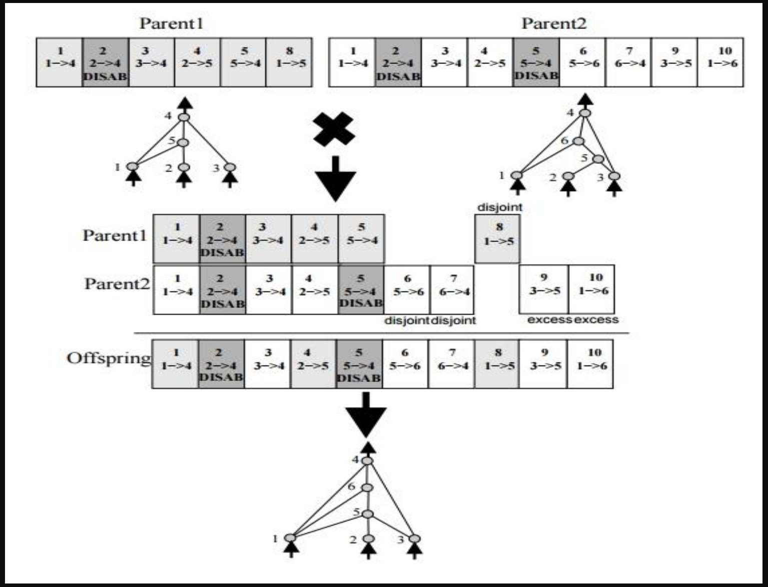
New fresh look!



Mutation types

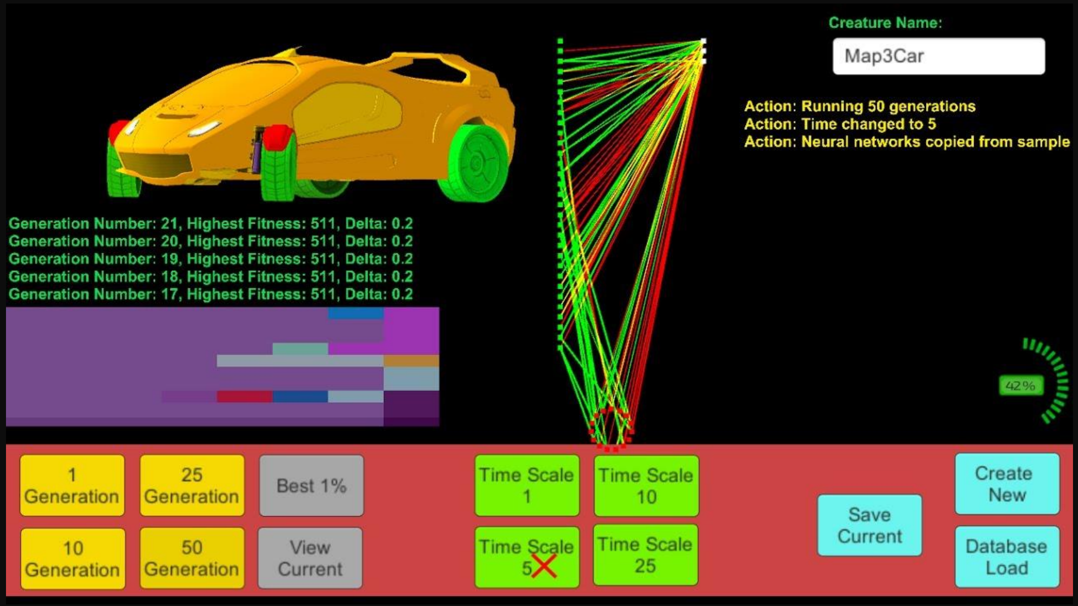
- Mutate with other network
- Mutate add node (with weight = 1)
- Mutate add connection (with weight = 1)
- Mutate activate/deactivate connection (percentage of all connections disabled)
- Mutate modify connection weight (multiply by a constant, usually ~0.9 – 1.1)
- Mutate shuffle connection weight (in range -1 – 1)

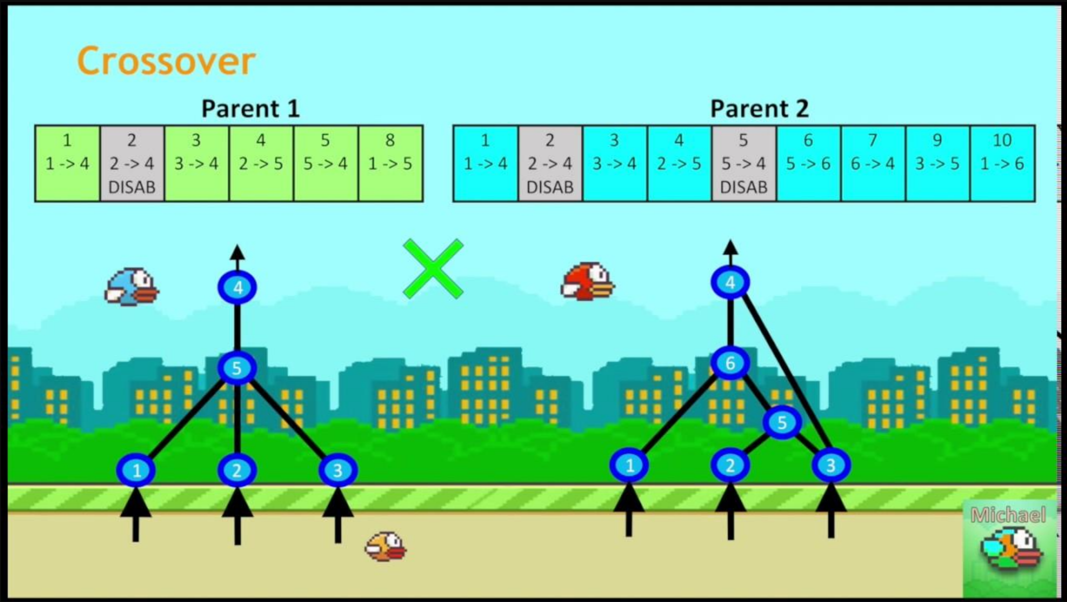


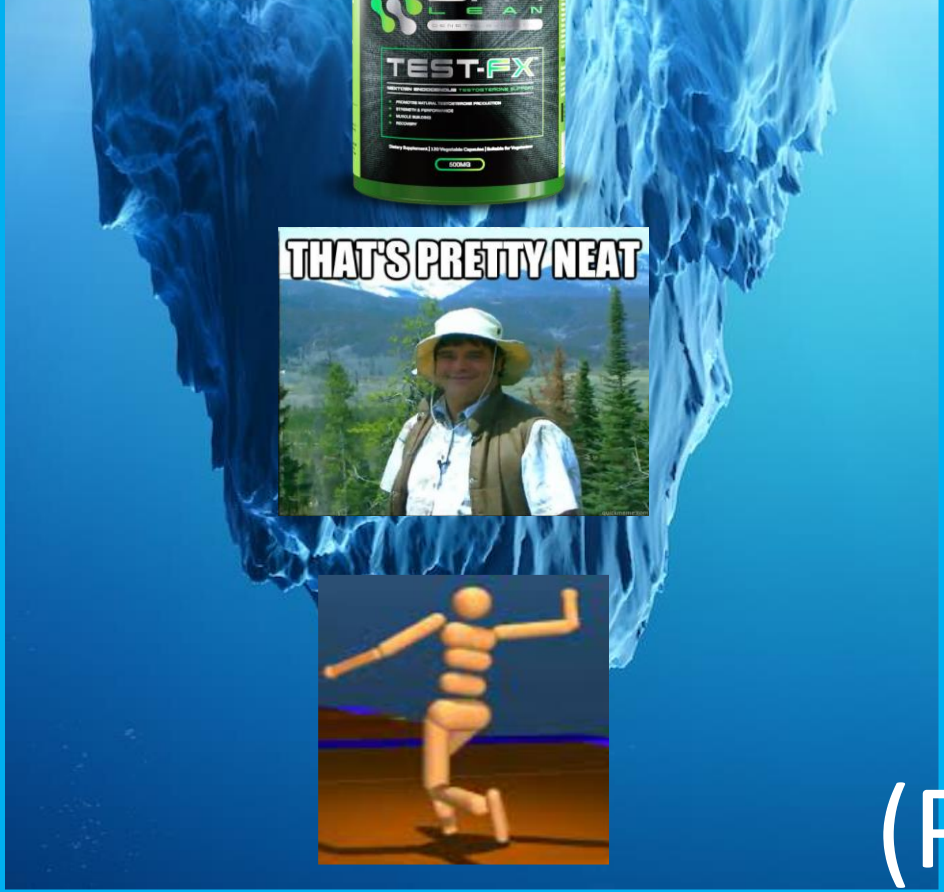


Benefits:
- Very advanced, can learn almost anything
- Looks badass
- Libraries can do most of the job
- Doesn’t overtrain (at least very difficult to do)
- Can make you $ if you upload it online for some sexy vids

Drawbacks:
- Training time is long
- Requires access to the game source code to amplify training time







PPO
(Proximal Policy Optimization)

Paper: https://arxiv.org/abs/1707.06347

How to design?

AI (Policy),
hence Proximal Policy
Optimization

Actor (Value Performer)

- Plays the game using observations – every frame it receives current game state, generates actions
- Every step it makes is saved to memory (action taken, log prob of action taken, rewards gathered per step, advantages – network loss)

Critic (Memory)

- Analyzes the actor's performance and adjusts policy, generates policy logits (values)
- Takes data from every training step (observation, action taken, log prob of action taken, GAE (decaying reward count)), self-optimizes

GAE – Generalized Advantage Estimation
https://arxiv.org/abs/1506.02438

Loss Functions

Actor (Value performer)

Critic (Memory)

- L2 loss (returns – values)^2
- Values – what the critic predicts
- Returns – Discounted rewards collected

- This unholy creation

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

Actor Computations

Actor (Value performer)

```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, dropout = 0.5):
        super().__init__()

        self.fc_1 = nn.Linear(input_dim, hidden_dim)
        self.fc_2 = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        x = self.fc_1(x)
        x = self.dropout(x)
        x = nn.relu(x)
        x = self.fc_2(x)
        return x
```

(actions) actions

Returns:

```
def calculate_returns(rewards, discount_factor, normalize = True):

    returns = []
    R = 0

    for r in reversed(rewards):
        R = r + R * discount_factor
        returns.insert(0, R)

    returns = torch.tensor(returns)

    if normalize:
        returns = (returns - returns.mean()) / returns.std()

    return returns
```

Advantages:

```
def calculate_advantages(returns, values, normalize = True):

    advantages = returns - values

    if normalize:

        advantages = (advantages - advantages.mean()) / advantages.s

    return advantages
```

Actor Training Loop

```
def train_actor():
    reward = 0
    simulation_memory = []
    for _ in range(nr_of_steps_per_simulation):
        observations = scene.get_next_state()
        actions = actor(observations)
        memory_value = critic(observations)
        actions_probabilities = softmax(actions)
        dist_actions = Categorical(actions_probabilities)
        action_taken = dist_actions.sample()
        log_prob_action = dist_actions.log_prob(action_taken)

        reward += scene.apply_action(action_taken)
        returns = calculate_returns(reward, discount_factor)

        gae = generalized_advantage_estimate = calculate_advantages(
                                                returns, memory_value)
        simulation_memory.append(
            [observations, action_taken, log_prob_action, returns, gae]
        )

        loss = (returns - memory_value)**2
        loss.backwards()
        adam_optimizer.step()
```

Critic Computations

Critic (Memory, Policy)

```python
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, dropout = 0.5):
        super().__init__()

        self.fc_1 = nn.Linear(input_dim, hidden_dim)
        self.fc_2 = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        x = self.fc_1(x)
        x = self.dropout(x)
        x = nn.relu(x)
        x = self.fc_2(x)
        return x
```

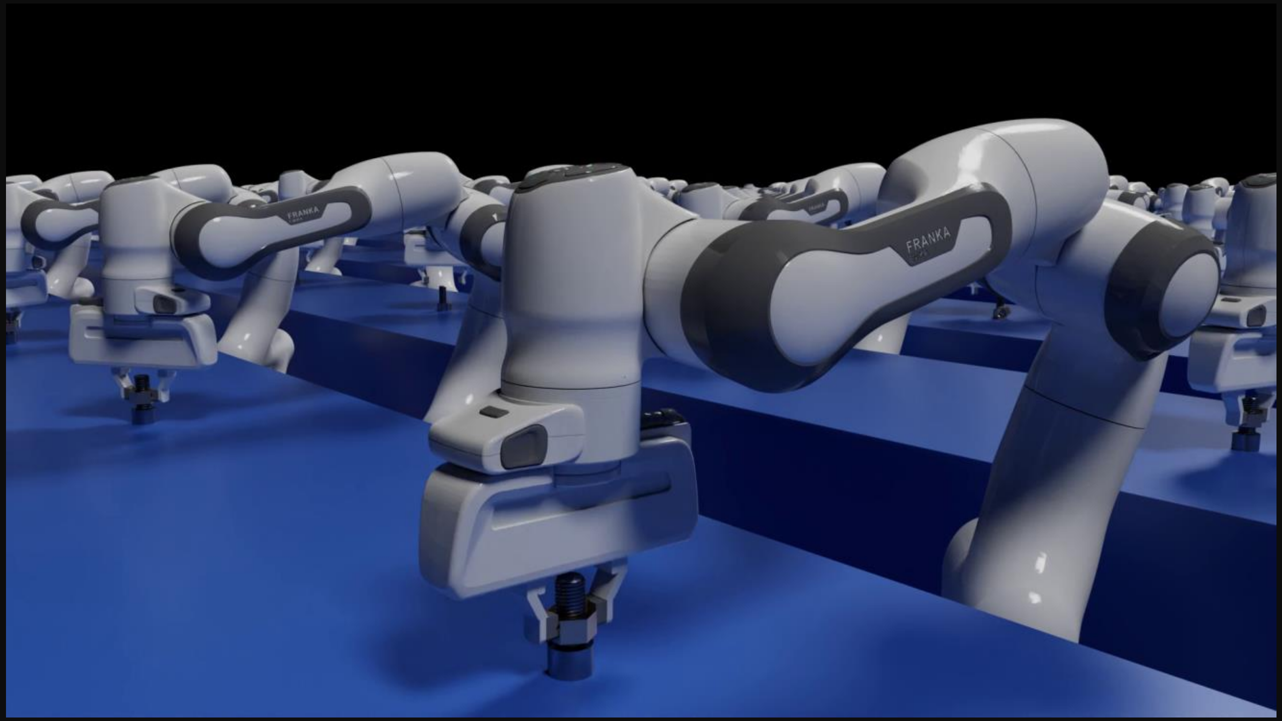
(1) value

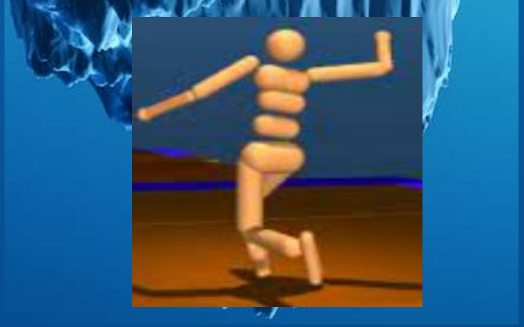
Loss function:

```python
def train_critic(old_log_prob_actions):
    observations, actions_taken, log_prob_actions, returns, gaes = zip(*simulation_memory)

    for i in range(max_number_of_critic_iterations):
        # Each time we are using ALL simulated frames
        policy_ratio = torch.exp(log_prob_actions - old_log_prob_actions) # division
        clipped_policy_ratio = policy_ratio.clamp(1 - clip_value, 1 + clip_value)
        policy_loss = policy_ratio * gaes
        clipped_policy_loss = clipped_policy_ratio * gaes

        final_loss = -torch.min(policy_loss, clipped_policy_loss).mean()
        final_loss.backwards()
        adam_optimizer.step()

        k1_div = (old_log_prob_actions - log_prob_actions).mean()
        if k1_div > threshold or k1_div < -threshold:
            break
```

Benefits:
- As advanced as it gets (as of 2017)
- Can learn to walk, punch, run, everything imaginable
- Algorithm is so complex you get hired for mid if you understand it

Drawbacks:
- Algorithm is so complex that you probably will not understand it
- Requires a lot of computing power





This is the End!

...

Agent57

Agent57 & How to code Minecraft AI

Coming soon...

Because training on laptops takes ages