AI in Games: Achievements and Challenges

Yuandong Tian
Facebook AI Research
AI works in a lot of situations

Object Recognition  Medical  Translation  Speech Recognition

Personalization  Surveillance  Smart Design  Board game
What AI still needs to improve

Exponential space to explore
Very few supervised data
Complicated/unknown environments with lots of corner cases.
Common Sense
The Charm of Games

Complicated long-term strategies.

Realistic Worlds
Game as a Vehicle of AI

- Infinite supply of fully labeled data
- Controllable and replicable
- Low cost per sample
- Faster than real-time
- Less safety and ethical concerns
- Complicated dynamics with simple rules.
Game as a Vehicle of AI

Algorithm is slow and data-inefficient

Require a lot of resources.

Abstract game to real-world

Hard to benchmark the progress
Game as a Vehicle of AI

Algorithm is slow and data-inefficient

Abstract game to real-world

Require a lot of resources.

Hard to benchmark the progress

Better Algorithm/System

Better Environment
Game as a Vehicle of AI

Algorithm is slow and data-inefficient

Abstract game to real-world

Require a lot of resources.

Hard to benchmark the progress

Better Algorithm/System

Better Environment
Our work

Better Environment

ELF: Extensive Lightweight and Flexible Framework
(Yuandong Tian et al, NIPS17)

House3D: An interactive 3D environment for navigation
(Yi Wu, Georgia Gkioxari, Yuxin Wu, Yuandong Tian)
Our work

Better Algorithm/System

DarkForest Go Engine
(Y. Tian, Y. Zhu, ICLR16)

Doom AI
(Yuxin Wu, Y. Tian, ICLR17)

MiniRTS
(Y. Tian, Q. Gong, W. Shang)
**ELF**: Extensive, Lightweight and Flexible Framework for Game Research

Yuandong Tian  
Qucheng Gong  
Wenling Shang  
Yuxin Wu  
Larry Zitnick

Facebook AI Research
Reinforcement Learning: Ideal and Reality

[R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction]
Reinforcement Learning: Ideal and Reality

Agent

State $S_t$

Reward $r_t$

Action $a_t$

Environment

State $S_{t+1}$

Reward $r_{t+1}$

Design Choices:

CPU, GPU?
Simulation, Replays
Concurrency

[R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction]
**ELF**: A simple for-loop

C++

Python

```python
while True:
    batched_states = GameContext.Wait()
    replies = model(batched_states)
    GameContext.Steps(replies)
```
ELF Characteristics

**Extensive**
Any games with C++ interfaces can be incorporated.

**Lightweight**
Fast. Mini-RTS (40K FPS per core)
Minimal resource usage (1GPU+several CPUs)
Fast training (half a day for a RTS game)

**Flexible**
Environment-Actor topology
Parametrized game environments.
Choice of different RL methods.
Extensibility

ELF

Go

ALE

Pong

Breakout

Mini-RTS

RTS Engine

Capture the Flag

Tower Defense
Lightweight

KFPS per CPU core for Pong (Atari)

- 1 core
- 2 cores
- 4 cores
- 8 cores
- 16 cores

OpenAI Gym
Flexibility

while True:
    batched = GameContext.Wait()
    replies = model(batched)
    GameContext.Steps(replies)
while True:
    ...
    if batch["type"] == "actor":
        ...
    elif batch["type"] == "train":
        ...
    Training

Flexibility
Flexibility

```
while True:
    ...
    if batch["type"] == "actor0":
        ...
    elif batch["type"] == "actor1":
        ...
```
Flexibility

```
while True:
    ...
    for i in range(n):
        if batch["type"] == "actor%d" % i:
            ...
```
Flexibility

Monte-Carlo Tree Search

while True:
    batched = GameContext.Wait()
    replies = model(batched)
    GameContext.Steps(replies)
ELF design

Produce (Games in C++)
ELF design

Game 1
- History buffer

Game 2
- History buffer

... 

Game N
- History buffer

Producer (Games in C++)
ELF design

Game 1

Game 2

Game N

History buffer

History buffer

History buffer

Collector

Producer (Games in C++)
ELF design

Producer (Games in C++)

Game 1

Game 2

Game N

Collector

Distributor

Batch with History info

Reply

Consumers (Python)
ELF design

Producer (Games in C++)

Game 1

Game 2

Game N

History buffer

History buffer

History buffer

Collector

Batch with History info

Distributor

Reply

Actor

Model

Consumers (Python)

A batch for actor
ELF design

Producer (Games in C++)

Game 1
Game 2
... 
Game N

History buffer
History buffer
... 
History buffer

Collector

Batch with History info

Distributor

A batch for actor

Actor

Model

Optimizer

Consumers (Python)
ELF design

Producer (Games in C++)

Game 1

Game 2

Game N

History buffer

Collector

Batch with History info

Distributor

Actor

Model

Optimizer

Consumers (Python)

Process

A batch for actor

A batch for optimizer

Reply

Batch with History info
Gorilla

Game → Actor → Model

Process

Game Experience

Replay Buffer

Model → Optimizer

Model → Optimizer

Model → Optimizer

Synchronization

Game Experience

Game Experience

Game Experience

[Nair et al, Massively Parallel Methods for Deep Reinforcement Learning, ICML 2015]
Asynchronized Advantageous Actor-Critic (A3C)

[Mnih et al, Asynchronous Methods for Deep Reinforcement Learning, ICML 2016]
[Babaeizadeh et al, Reinforcement Learning through Asynchronous Advantage Actor–Critic on a GPU, ICLR 2017]
ELF: A unified framework

Off-policy training
Deep Q-learning

One-to-One
Vanilla A3C

Many-to-One
BatchA2C, GA3C
ELF: A unified framework

- **Off-policy training**
  - Deep Q-learning

- **One-to-One**
  - Vanilla A3C

- **Many-to-One**
  - BatchA3C, GA3C

- **One-to-Many**
  - Self-Play, Monte-Carlo Tree Search
Open Source

https://github.com/facebookresearch/ELF
**MiniRTS**: A miniature RTS engine

---

**Platform** | Frame per second
--- | ---
ALE | 6,000
Open AI Universe | 60
Malmo | 120
DeepMind Lab | 287*/866**
VizDoom | 7,000
TorchCraft | 2,000
**MiniRTS** | 40,000

* Using CPU only | ** Using CPUs and GPU
# MiniRTS

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base</strong></td>
<td>Build workers and collect resources.</td>
</tr>
<tr>
<td><strong>Resource</strong></td>
<td>Contains 1000 minerals.</td>
</tr>
<tr>
<td><strong>Barracks</strong></td>
<td>Build melee attacker and range attacker.</td>
</tr>
<tr>
<td><strong>Worker</strong></td>
<td>Build barracks and gather resource.</td>
</tr>
<tr>
<td></td>
<td>Low speed in movement and low attack damage.</td>
</tr>
<tr>
<td><strong>Melee Tank</strong></td>
<td>High HP, medium movement speed, short attack range, high attack damage.</td>
</tr>
<tr>
<td><strong>Range Tank</strong></td>
<td>Low HP, high movement speed, long attack range and medium attack damage.</td>
</tr>
</tbody>
</table>
Using Internal Game data and Actor-Critic Models. Reward is only available once the game is over.
9 Discrete Strategic Actions

<table>
<thead>
<tr>
<th>No.</th>
<th>Action name</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IDLE</td>
<td>Do nothing</td>
</tr>
<tr>
<td>2</td>
<td>BUILD WORKER</td>
<td>If the base is idle, build a worker</td>
</tr>
<tr>
<td>3</td>
<td>BUILD BARRACK</td>
<td>Move a worker (gathering or idle) to an empty place and build a barrack.</td>
</tr>
<tr>
<td>4</td>
<td>BUILD MELEE ATTACKER</td>
<td>If we have an idle barrack, build an melee attacker.</td>
</tr>
<tr>
<td>5</td>
<td>BUILD RANGE ATTACKER</td>
<td>If we have an idle barrack, build a range attacker.</td>
</tr>
<tr>
<td>6</td>
<td>HIT AND RUN</td>
<td>If we have range attackers, move towards opponent base and attack. Take advantage of their long attack range and high movement speed to hit and run if enemy counter-attack.</td>
</tr>
<tr>
<td>7</td>
<td>ATTACK</td>
<td>All melee and range attackers attack the opponent’s base.</td>
</tr>
<tr>
<td>8</td>
<td>ATTACK IN RANGE</td>
<td>All melee and range attackers attack enemies in sight.</td>
</tr>
<tr>
<td>9</td>
<td>ALL DEFEND</td>
<td>All troops attack enemy troops near the base and resource.</td>
</tr>
</tbody>
</table>
Rule-based AIs

AI_SIMPLE
Build 5 tanks and attack

AI_HIT_AND_RUN
Build 2 tanks and harass

*MiniRTS trains with a single GPU and 6 CPUs in half a day.*
## Win rate against rule-based AI

### Network Architecture

```
Conv → BN → ReLU
```

<table>
<thead>
<tr>
<th>Win Rate (10K games)</th>
<th>SIMPLE (median)</th>
<th>SIMPLE (mean/std)</th>
<th>HIT_AND_RUN (median)</th>
<th>HIT_AND_RUN (mean/std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReLU</td>
<td>52.8</td>
<td>54.7 (±4.2)</td>
<td>60.4</td>
<td>57.0 (±6.8)</td>
</tr>
<tr>
<td>Leaky ReLU</td>
<td>59.8</td>
<td>61.0 (±2.6)</td>
<td>60.2</td>
<td>60.3 (±3.3)</td>
</tr>
<tr>
<td>ReLU + BN</td>
<td>61.0</td>
<td>64.4 (±7.4)</td>
<td>55.6</td>
<td>57.5 (±6.8)</td>
</tr>
<tr>
<td>Leaky ReLU + BN</td>
<td><strong>72.2</strong></td>
<td><strong>68.4 (±4.3)</strong></td>
<td><strong>65.5</strong></td>
<td><strong>63.6 (±7.9)</strong></td>
</tr>
</tbody>
</table>
Curriculum Training

<table>
<thead>
<tr>
<th>Win Rate</th>
<th>Without curriculum training</th>
<th>With curriculum training</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI_SIMPLE</td>
<td>66.0 (±2.4)</td>
<td>68.4 (±4.3)</td>
</tr>
<tr>
<td>AI_HIT_AND_RUN</td>
<td>54.4 (±15.9)</td>
<td>63.6 (±7.9)</td>
</tr>
</tbody>
</table>

First $k$ decisions made by AI_SIMPLE then made by trained AI

\[ k \sim \text{Uniform}[0, K] \]
\[ K \propto \beta^{-\text{#game\_played}} \]
## Transfer Learning

<table>
<thead>
<tr>
<th>Win Rate</th>
<th>AI_SIMPLE</th>
<th>AI_HIT_AND_RUN</th>
<th>Combined (50%SIMPLE+50% H&amp;R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td><strong>68.4 (±4.3)</strong></td>
<td>26.6(±7.6)</td>
<td>47.5(±5.1)</td>
</tr>
<tr>
<td>HIT_AND_RUN</td>
<td>34.6(±13.1)</td>
<td><strong>63.6 (±7.9)</strong></td>
<td>49.1(±10.5)</td>
</tr>
<tr>
<td>Combined</td>
<td>51.8(±10.6)</td>
<td>54.7(±11.2)</td>
<td><strong>53.2(±8.5)</strong></td>
</tr>
</tbody>
</table>
Monte Carlo Tree Search

<table>
<thead>
<tr>
<th>Win Rate</th>
<th>AI_SIMPLE</th>
<th>AI_HIT_AND_RUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>24.2 (±3.9)</td>
<td>25.9 (±0.6)</td>
</tr>
<tr>
<td>MCTS*</td>
<td>73.2 (±0.6)</td>
<td>62.7 (±2.0)</td>
</tr>
<tr>
<td>Trained AI</td>
<td>68.4 (±4.3)</td>
<td>63.6 (±7.9)</td>
</tr>
</tbody>
</table>

* repeat on 1000 games, each using 800 rollouts.

MCTS uses complete information and perfect dynamics.
Recent Update

Win rate versus iterations

<table>
<thead>
<tr>
<th>Method</th>
<th>Vanilla</th>
<th>Vanilla (hist=4)</th>
<th>RNN</th>
<th>BuildHistory</th>
<th>PrevSeen</th>
<th>Complete Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win rate</td>
<td>72.9±1.8</td>
<td>79.8±0.7</td>
<td>79.7±1.3</td>
<td>80.8±1.7</td>
<td>81.4±0.8</td>
<td>81.7±0.7</td>
</tr>
</tbody>
</table>
Ongoing Work

Engineering

• Richer game scenarios for MiniRTS.
  • LUA scripting support
  • Multiple bases (Expand? Rush? Defending?)
  • More complicated units.

• Realistic action space
  • One command per unit

Research

• Model-based Reinforcement Learning
• Hierarchical RL
• Self-Play (Trained AI versus Trained AI)
LUA Interface for MiniRTS

- Easy to change game dynamics
  - Don’t need to touch C++.
- Comparable speed to C++
  - 1.5x slower than compiled code.

```lisp
(g_funcs = {})

  function g_funcs.attack(env, cmd)
    local target = env:unit(cmd.target)
    local u = env:self()
    if target:isdead() or not u:canSee(target) then
      -- c_print("Task finished!")
      return global.CMD_COMPLETE
    end
    local att_r = u:att_r()
    local in_range = env:dist_sqr(target:p()) <= att_r * att_r
    if u:cd_expired(global.CD_ATTACK) and in_range then
      -- print("Attacking .. ")
      -- Then we need to attack.
      if att_r <= 1.0 then
        env:send_cmd_melee_attack(cmd.target, u:att())
      else
        env:send_cmd_emitter_bullet(cmd.target, u:att())
      end
      env:cd_start(global.CD_ATTACK)
    else
      if not in_range then
        -- print("Moving towards target .. ")
        env:move_towards(target)
      end
    end
  end
```
# A3C

def update(self, batch):
    ''' Actor critic model '''
    R = deepcopy(batch["V"][T - 1])
    batchsize = R.size(0)
    R.resize_(batchsize, 1)

    for t in range(T - 2, -1, -1):
        # Forward pass
        curr = self.model_interface.forward("model", batch.hist(t))

        # Compute the reward.
        R = R * self.args.discount + batch["r"][t]
        # If we see any terminal signal, do not backprop
        for i, terminal in enumerate(batch["terminal"][t]):
            if terminal:
                R[t][i] = curr["V"].data[i]

        # We need to set it beforehand.
        self.policy_gradient_weights = R - curr["V"].data

        # Compute policy gradient error:
        errs = self._compute_policy_entropy_err(curr["pi"], batch["a"][t])
        # Compute critic error
        value_err = self.value_loss(curr["V"], Variable(R))

        overall_err = value_err + errs["policy_err"]
        overall_err += errs["entropy_err"] * self.args.entropy_ratio
        overall_err.backward()
House3D: A rich and realistic 3D environment

[Yi Wu et al, Building Generalizable Agents with a Realistic and Rich 3D Environment, ICLR 2018 submission]
SUNCG Dataset

SUNCG dataset, 45K scenes, all objects are fully labeled.
Multi-modality

Depth

Segmentation mask

RGB image

Top-down map
Architecture
## Comparison

<table>
<thead>
<tr>
<th>Environment</th>
<th>3D</th>
<th>Realistic</th>
<th>Large-scale</th>
<th>Fast-speed</th>
<th>Customizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atari (Bellemare et al., 2013)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>OpenAI Universe (Shi et al., 2017)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>Malmo (Johnson et al., 2016)</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>DeepMind Lab (Beattie et al., 2016)</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>VizDoom (Kempka et al., 2016)</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>AI2-THOR (Zhu et al., 2017)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>House3D</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Successful Rate

(a) Training performances

(b) Generalization performances on the test set
Videos
DarkForest: Go engine
Yuandong Tian and Yan Zhu, ICLR 2016

• DCNN as a tree policy
  • Predict next k moves (rather than next move)
  • Trained on 170k KGS dataset/80k GoGoD, **57.1%** accuracy.
  • KGS 3D without search (0.1s per move)
  • Release 3 month before AlphaGo, < 1% GPUs (from Aja Huang)
Our computer Go player: DarkForest

| Name                                                      |
|------------------------------------------------**********|
| Our/enemy liberties                                      |
| Ko location                                              |
| Our/enemy stones/empty place                             |
| Our/enemy stone history                                  |
| Opponent rank                                            |

Feature used for DCNN

feature type: standard

![Graph showing winrate against Pachi 10k over epochs for different nstep values.](image)
Pure DCNN

*darkforest*: Only use top-1 prediction, trained on KGS
*darkfores1*: Use top-3 prediction, trained on GoGoD
*darkfores2*: *darkfores1* with fine-tuning.

<table>
<thead>
<tr>
<th></th>
<th>GnuGo (level 10)</th>
<th>Pachi 10k</th>
<th>Pachi 100k</th>
<th>Fuego 10k</th>
<th>Fuego 100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clark &amp; Storkey (2015)</td>
<td>91.0</td>
<td>-</td>
<td>-</td>
<td>14.0</td>
<td></td>
</tr>
<tr>
<td>Maddison et al. (2015)</td>
<td>97.2</td>
<td>47.4</td>
<td>11.0</td>
<td>23.3</td>
<td>12.5</td>
</tr>
<tr>
<td>darkforest</td>
<td>98.0 ± 1.0</td>
<td>71.5 ± 2.1</td>
<td>27.3 ± 3.0</td>
<td>84.5 ± 1.5</td>
<td>56.7 ± 2.5</td>
</tr>
<tr>
<td>darkfores1</td>
<td>99.7 ± 0.3</td>
<td>88.7 ± 2.1</td>
<td>59.0 ± 3.3</td>
<td>93.2 ± 1.5</td>
<td>78.0 ± 1.7</td>
</tr>
<tr>
<td>darkfores2</td>
<td>100 ± 0.0</td>
<td>94.3 ± 1.7</td>
<td>72.6 ± 1.9</td>
<td>98.5 ± 0.1</td>
<td>89.7 ± 2.1</td>
</tr>
</tbody>
</table>

Win rate between DCNN and open source engines.
Monte Carlo Tree Search

Aggregate win rates, and search towards the good nodes.

(a) 

(b) 

(c) 

Tree policy
Default policy
DCNN + MCTS

*darkfmcts3:* Top-3/5, 75k rollouts, ~12sec/move, KGS 5d

<table>
<thead>
<tr>
<th></th>
<th>darkforest+MCTS</th>
<th>darkfores1+MCTS</th>
<th>darkfores2+MCTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vs pure DCNN (1000rl/top-20)</td>
<td>84.8%</td>
<td>74.0%</td>
<td>62.8%</td>
</tr>
<tr>
<td>Vs pure DCNN (1000rl/top-5)</td>
<td>89.6%</td>
<td>76.4%</td>
<td>68.4%</td>
</tr>
<tr>
<td>Vs pure DCNN (1000rl/top-3)</td>
<td>91.6%</td>
<td>89.6%</td>
<td><strong>79.2%</strong></td>
</tr>
<tr>
<td>Vs pure DCNN (5000rl/top-5)</td>
<td>96.8%</td>
<td>94.3%</td>
<td>82.3%</td>
</tr>
<tr>
<td>Vs Pachi 10k (pure DCNN baseline)</td>
<td>71.5%</td>
<td>88.7%</td>
<td>94.3%</td>
</tr>
<tr>
<td>Vs Pachi 10k (1000rl/top-20)</td>
<td>91.2% (+19.7%)</td>
<td>92.0% (+3.3%)</td>
<td>95.2% (+0.9%)</td>
</tr>
<tr>
<td>Vs Pachi 10k (1000rl/top-5)</td>
<td>88.4% (+16.9%)</td>
<td>94.4% (+5.7%)</td>
<td>97.6% (+3.3%)</td>
</tr>
<tr>
<td>Vs Pachi 10k (1000rl/top-3)</td>
<td>95.2% (+23.7%)</td>
<td>98.4% (+9.7%)</td>
<td>99.2% (+4.9%)</td>
</tr>
<tr>
<td>Vs Pachi 10k (5000/top-5)</td>
<td>98.4%</td>
<td>99.6%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

Win rate between DCNN + MCTS and open source engines.
DarkForest

• DCNN+MCTS
  • Use top3/5 moves from DCNN, 75k rollouts.
  • Stable KGS 5d. Open source.  https://github.com/facebookresearch/darkforestGo
  • 3rd place on KGS January Tournaments
  • 2nd place in 9th UEC Computer Go Competition (Not this time 😊)

DarkForest versus Koichi Kobayashi (9p)
Win Rate analysis (using DarkForest) (AlphaGo versus Lee Sedol)

New version of DarkForest on ELF platform

https://github.com/facebookresearch/ELF/tree/master/go
First Person Shooter (FPS) Game

Yuxin Wu, Yuandong Tian, ICLR 2017

Play the game from the raw image!
Network Structure

Simple Frame Stacking is very useful (rather than Using LSTM)
Actor-Critic Models

\[ \nabla \log \pi(a|s_t)(R_t - V(s_t)) \]

Update Policy network

Reward

Update Value network

\[ (R_t - V(s_t)) \nabla V(s_t) \]

Encourage actions leading to states with high-than-expected value.
Encourage value function to converge to the true cumulative rewards.
Keep the diversity of actions
Curriculum Training

From simple to complicated

FlatMap

CIGTrack1
Curriculum Training

<table>
<thead>
<tr>
<th></th>
<th>Class 0</th>
<th>Class 1</th>
<th>Class 2</th>
<th>Class 3</th>
<th>Class 4</th>
<th>Class 5</th>
<th>Class 6</th>
<th>Class 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>0.8</td>
<td>1.0</td>
</tr>
<tr>
<td>Health</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>
VizDoom AI Competition 2016 (Track1)

We won the first place!

<table>
<thead>
<tr>
<th>Rank</th>
<th>Bot</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>Total frags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F1</td>
<td>56</td>
<td>62</td>
<td>n/a</td>
<td>54</td>
<td>47</td>
<td>43</td>
<td>47</td>
<td>55</td>
<td>50</td>
<td>48</td>
<td>50</td>
<td>559</td>
</tr>
<tr>
<td>2</td>
<td>Arnold</td>
<td>36</td>
<td>34</td>
<td>42</td>
<td>36</td>
<td>36</td>
<td>45</td>
<td>36</td>
<td>39</td>
<td>n/a</td>
<td>33</td>
<td>36</td>
<td>413</td>
</tr>
<tr>
<td>3</td>
<td>CLYDE</td>
<td>37</td>
<td>n/a</td>
<td>38</td>
<td>32</td>
<td>37</td>
<td>30</td>
<td>46</td>
<td>42</td>
<td>33</td>
<td>24</td>
<td>44</td>
<td>393</td>
</tr>
</tbody>
</table>

Videos:
https://www.youtube.com/watch?v=94EPSjQH38Y
https://www.youtube.com/watch?v=Qv4esGWOG7w&t=394s
Visualization of Value functions

Best 4 frames (agent is about to shoot the enemy)

Worst 4 frames (agent missed the shoot and is out of ammo)
Thanks!