DeepMind

Introduction to OpenSpiel

Marc Lanctot

Joint work with Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, Jonah Ryan-Davis, and several external contributors!

Many, many great collaborators!

Intro to OpenSpiel (Released Aug '19)

- Open source framework for research on RL, search, and planning in games
- Main impl in C++ and Python. Also:
  - Swift
  - Julia (contributed post-release)
- > 25 games
- > 10 algorithms

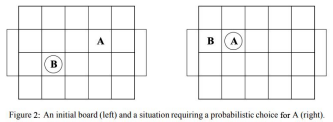
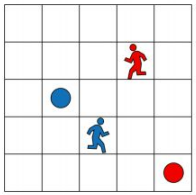
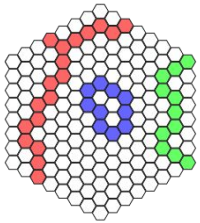
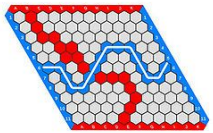


Figure 2: An initial board (left) and a situation requiring a probabilistic choice for A (right).







OpenSpiel

Supports:

- n-player games
- Zero-sum, coop, general-sum
- Perfect / imperfect info
- Simultaneous-move games

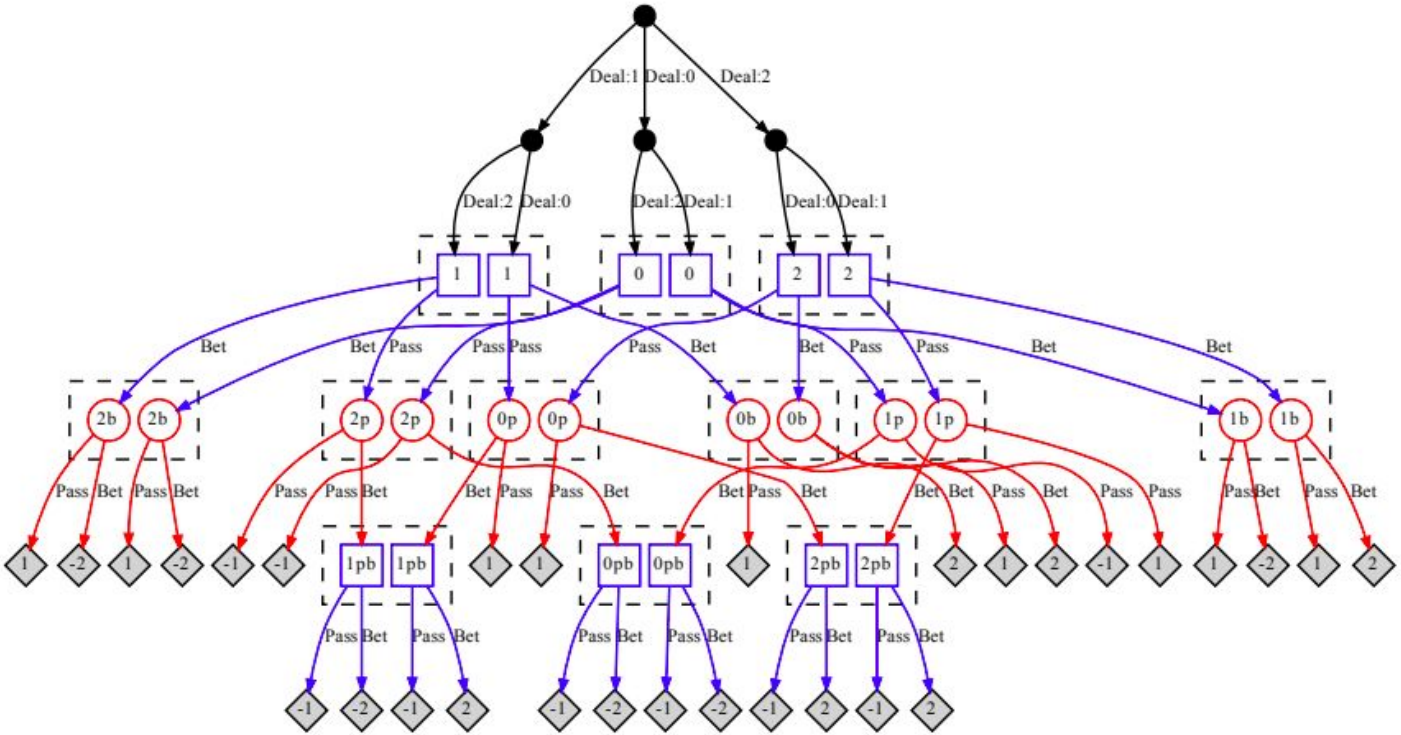
Tour of OpenSpiel

Main web site: github.com/deepmind/open_spiel/

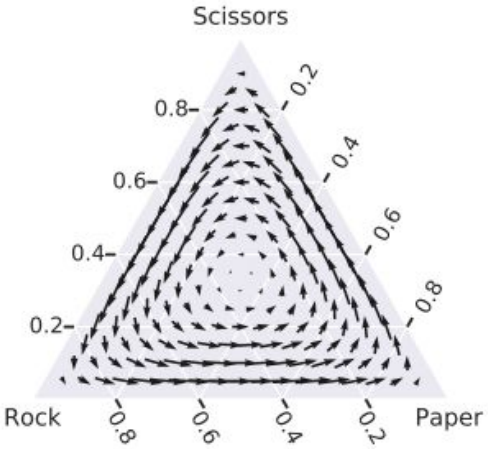
(Link to open colab on the main site)

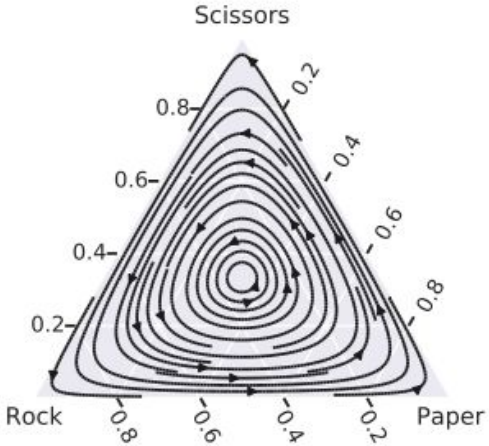
- Contributors
- Games
- Algorithms

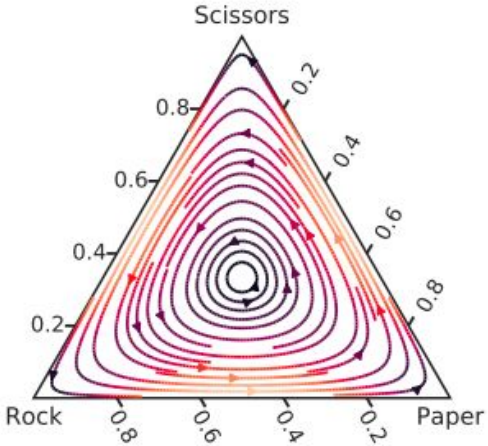
OpenSpiel: Example Viz (Kuhn Poker)



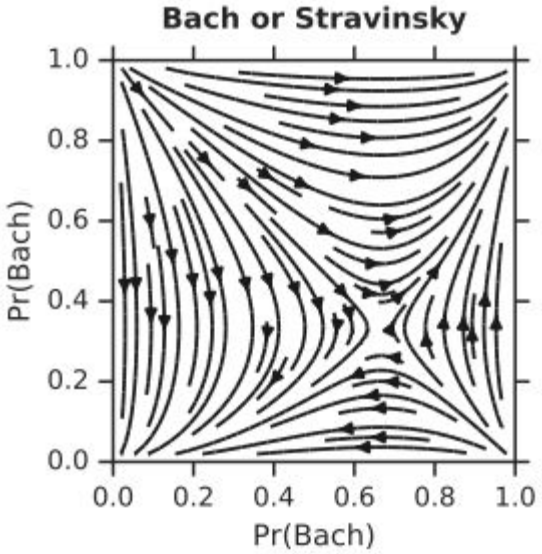
OpenSpiel: Example Viz (Replicator dynamics)

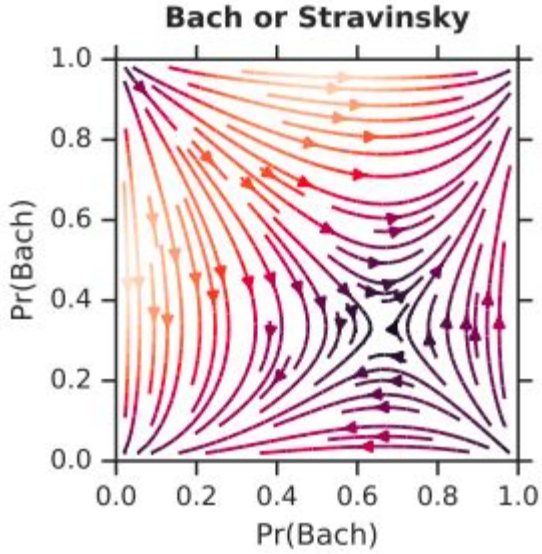






OpenSpiel: Example Viz (Replicator dynamics)





Motivation: Why another games / RL library?

1. Promote work on **general** multiagent RL
   a. “Atari Learning Environment” of multiagent/games
   b. General game-**learning**
2. **Games** have specific requirements and use cases:
   a. Illegal moves, turn-based, etc.
3. Connecting research communities!
4. Open code, metrics, communication, progress
5. Reproducibility in research

OpenSpiel: Design & Code

Design Philosophy

1. **Keep it simple.**
2. **Keep it light.**

Main structure:

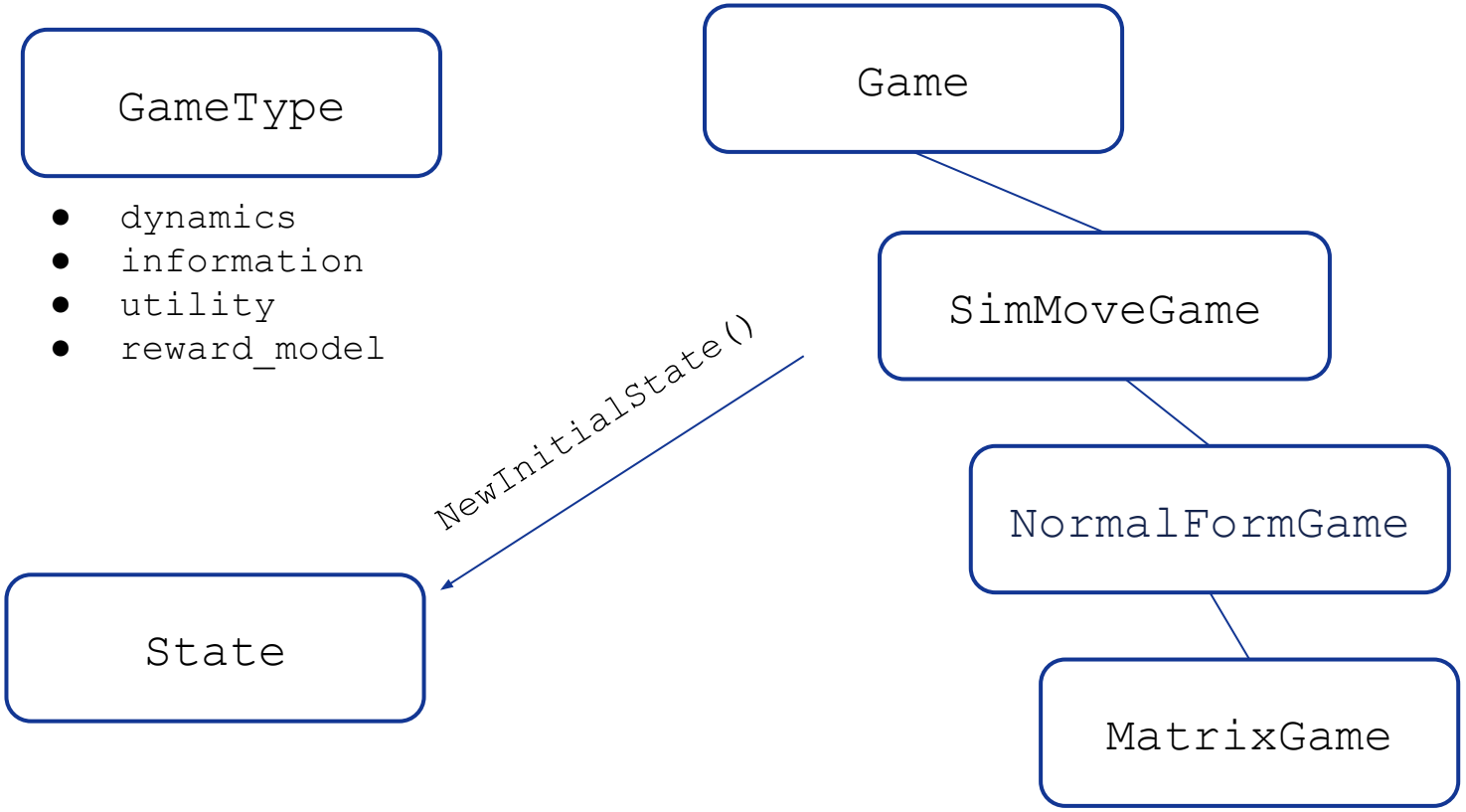
- C++ core + Python API
- Swift port
- Julia API
- Go API (in the works)
- Games in C++
- Algs in C++ and Python
- Many examples / colab

Example

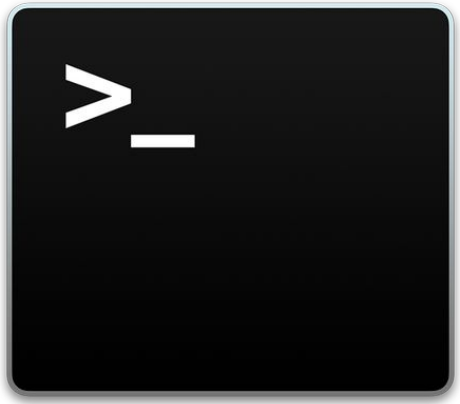
```python
import random
import pyspiel
import numpy as np

game = pyspiel.load_game("kuhn_poker")
state = game.new_initial_state()
while not state.is_terminal():
  legal_actions = state.legal_actions()
  if state.is_chance_node():
    # Sample a chance event outcome.
    outcomes_with_probs = state.chance_outcomes()
    action_list, prob_list = zip(*outcomes_with_probs)
    action = np.random.choice(action_list, p=prob_list)
    state.apply_action(action)
  else:
    # The algorithm can pick an action based on an observation (fully observable
    # games) or an information state (information available for that player)
    # We arbitrarily select the first available action as an example.
    action = legal_actions[0]
    state.apply_action(action)
```

Object-Oriented API



OpenSpiel Live Demo, Part 1



- Showcase basic OpenSpiel core API by example via python interpreter.
- Feel free to follow along in colab or locally!
- Transcript of demo: `demo1.txt`

Multiagent Learning Dynamics

Nash Convergence of Gradient Dynamics in General-Sum Games

Satinder Singh
AT&T Labs
Florham Park, NJ 07932
baveja@research.att.com

Michael Kearns
AT&T Labs
Florham Park, NJ 07932
mkearns@research.att.com

Yishay Mansour
Tel Aviv University
Tel Aviv, Israel
mansour@math.tau.ac.il

Singh, Kearns & Mansour '03, Infinitesimal Gradient Ascent (IGA)

Multiagent Learning Dynamics

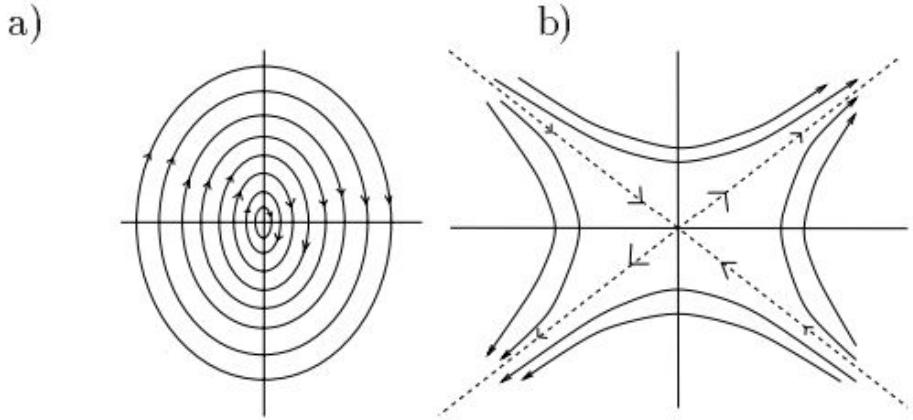


Figure 1: The general form of the dynamics: a) when U has imaginary eigenvalues and b) when U has real eigenvalues.

Image from Singh, Kearns, & Mansour '03

Formalize optimization as a dynamical system:

policy gradients

Analyze using well-established techniques

Replicator Dynamics

→ Evolutionary Game Theory: **replicator dynamics**

$$\dot{\pi}_t(a) = \pi_t(a)\left[u(a, \boldsymbol{\pi}_t) - \bar{u}(\boldsymbol{\pi}_t)\right]$$

time derivative

Replicator Dynamics

→ Evolutionary Game Theory: **replicator dynamics**

$$\dot{\pi}_t(a) = \pi_t(a)\left[u(a, \boldsymbol{\pi}_t) - \bar{u}(\boldsymbol{\pi}_t)\right]$$

time derivative

utility of action a against the joint policy / population of other players

Replicator Dynamics

→ Evolutionary Game Theory: **replicator dynamics**

$$\dot{\pi}_t(a) = \pi_t(a)\left[u(a, \boldsymbol{\pi}_t) - \bar{u}(\boldsymbol{\pi}_t)\right]$$

time derivative

utility of action a against the joint policy / population of other players

Expected / average utility of the joint policy / population

Phase Portraits

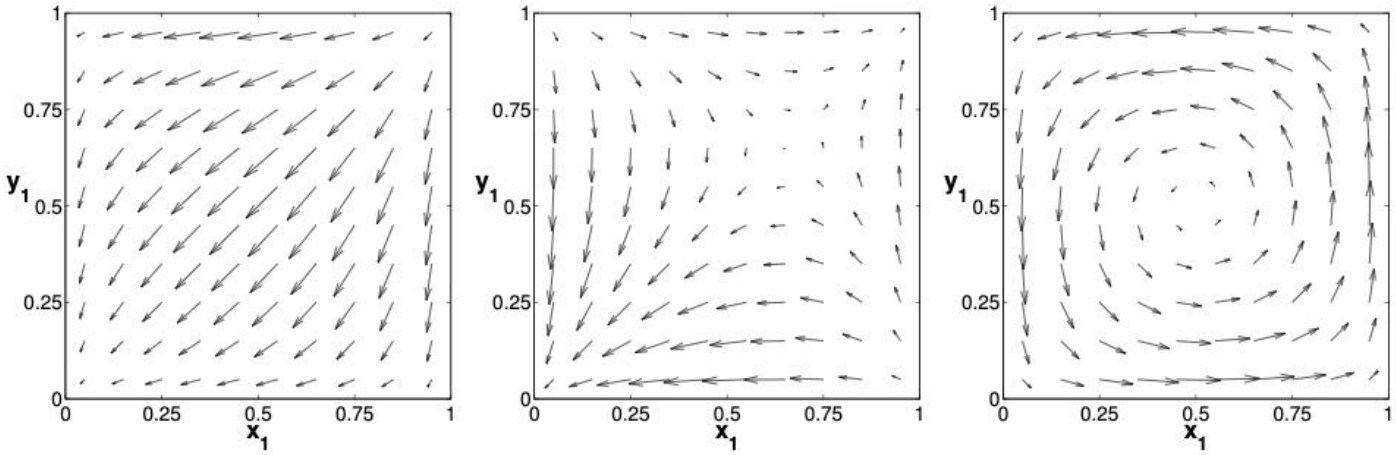
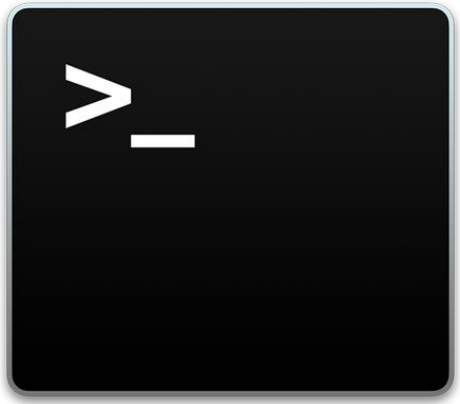


Figure 4: The replicator dynamics, plotted in the unit simplex, for the prisoner's dilemma (left), the stag hunt (center), and matching pennies (right).

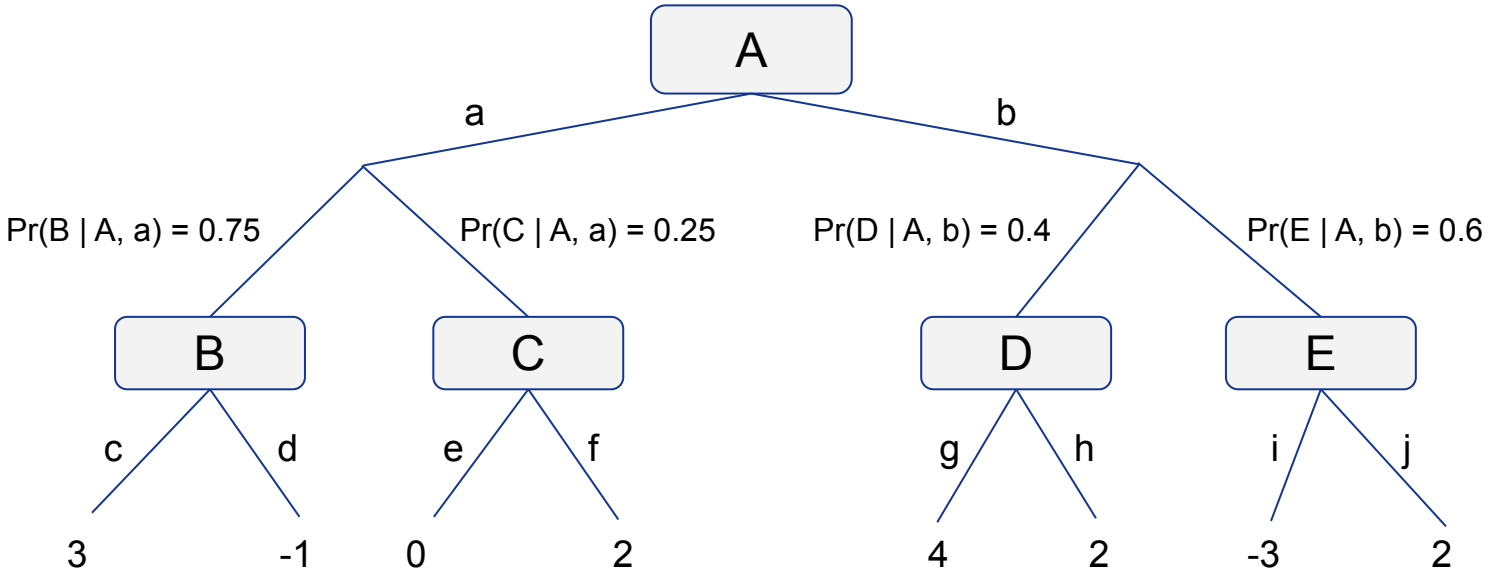
[Bloembergen et al. 2015]

OpenSpiel Live Demo, Part 2

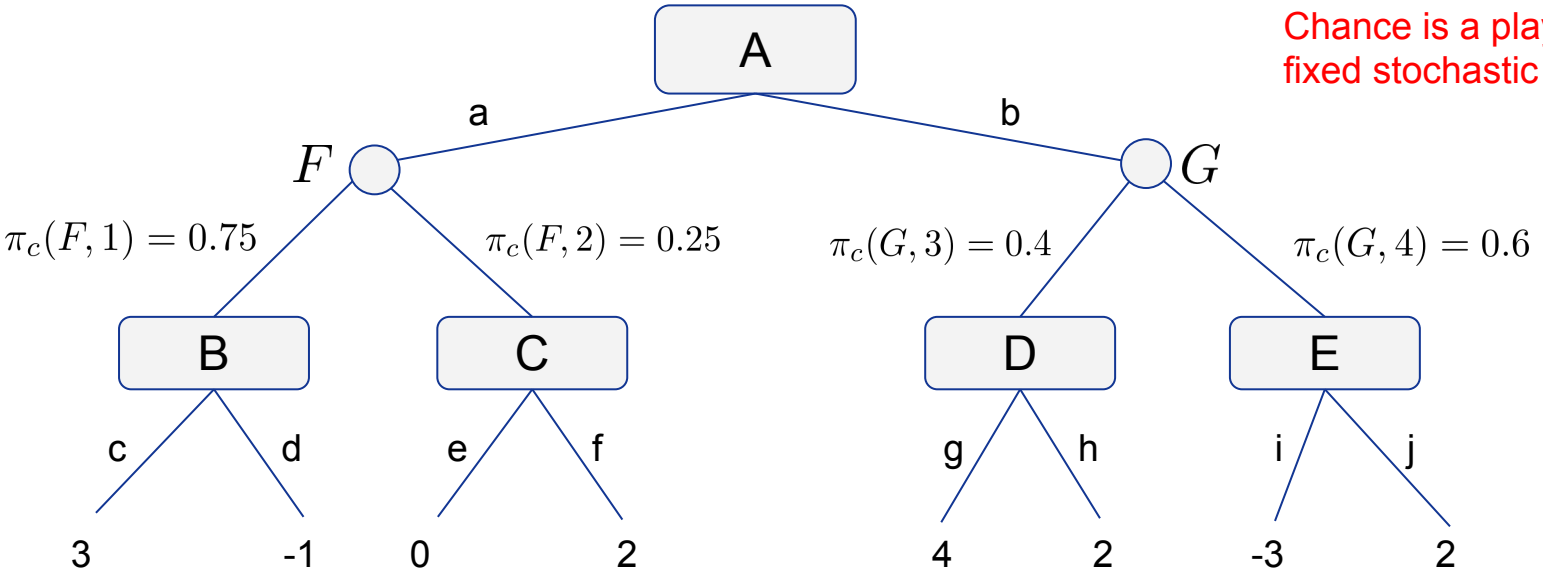


- Matrix games and learning dynamics
- Feel free to follow along in colab or locally!
- Transcript of demo: `demo2.txt`

A simple MDP

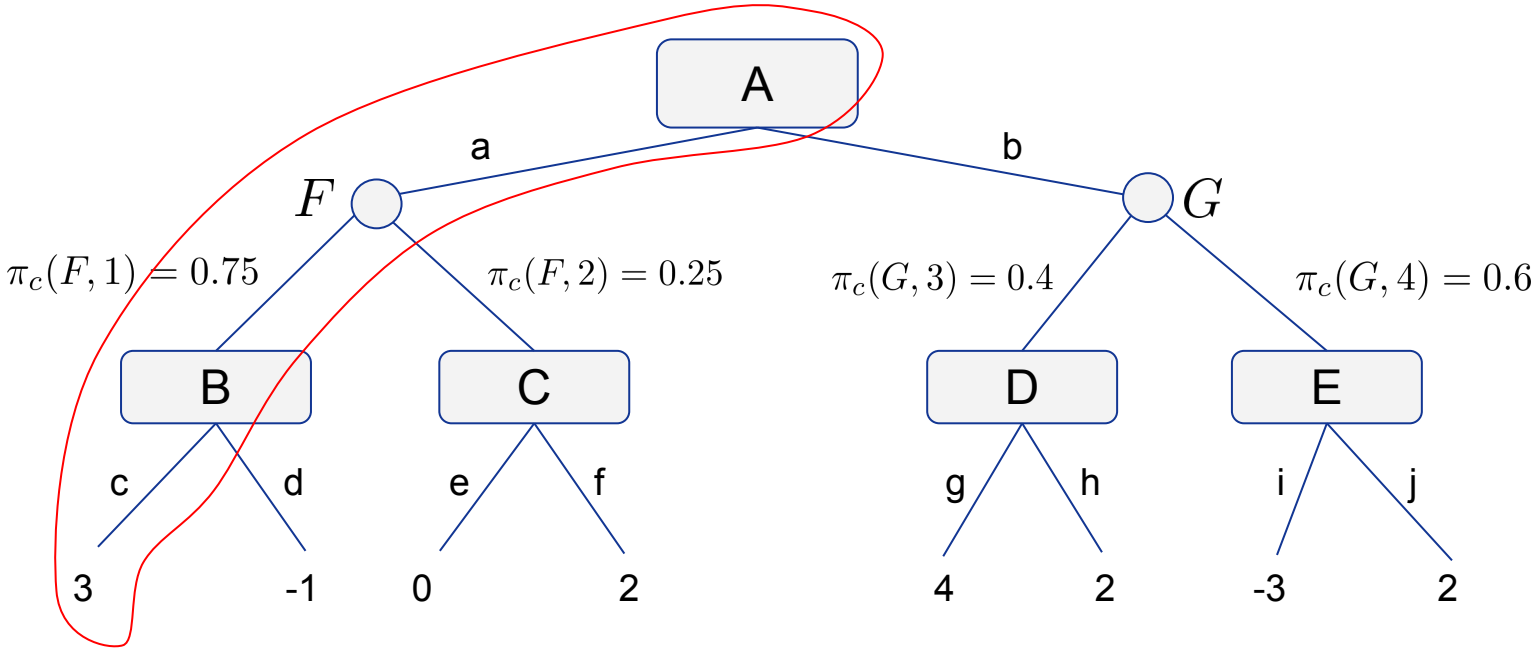

A
a
b
Pr(B | A, a) = 0.75
Pr(C | A, a) = 0.25
Pr(D | A, b) = 0.4
Pr(E | A, b) = 0.6
B
C
D
E
c
d
e
f
g
h
i
j
3
-1
0
2
4
2
-3
2

A simple ~~MDP~~ Multiagent System



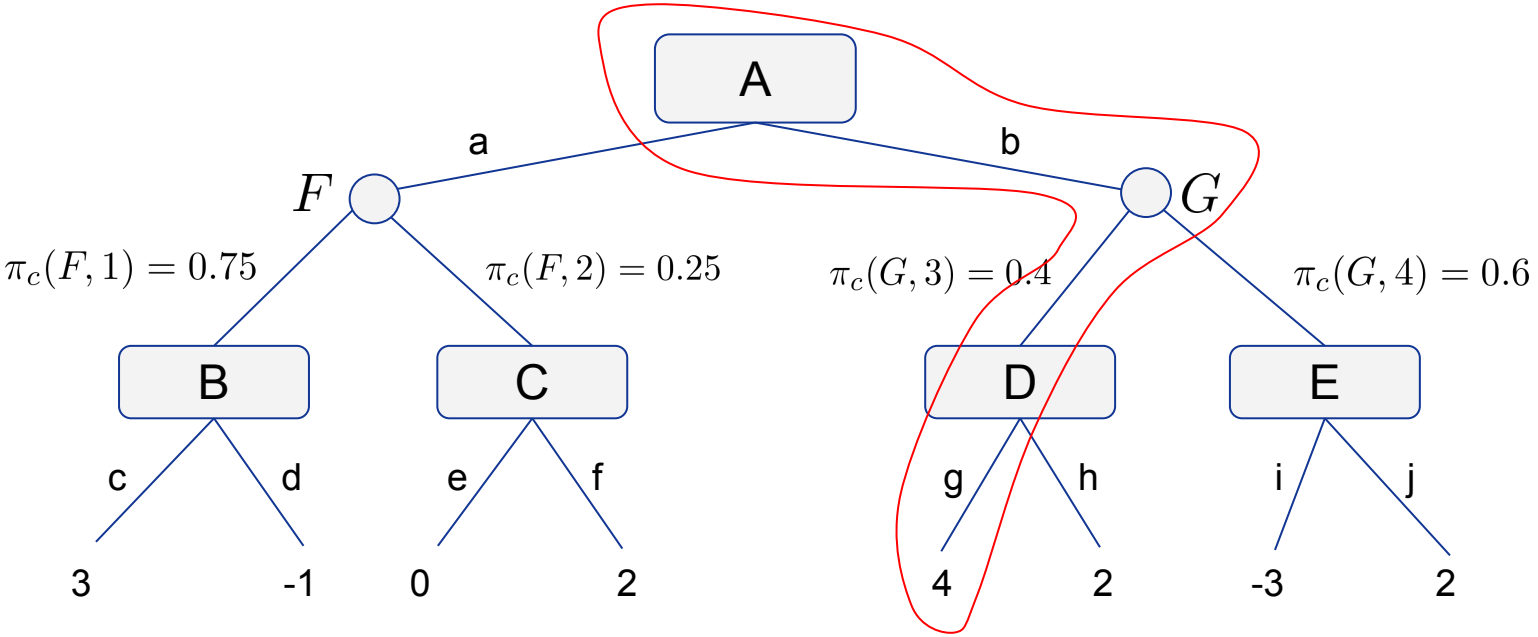
Chance is a player with a fixed stochastic policy!

Terminal history A.K.A. Episode



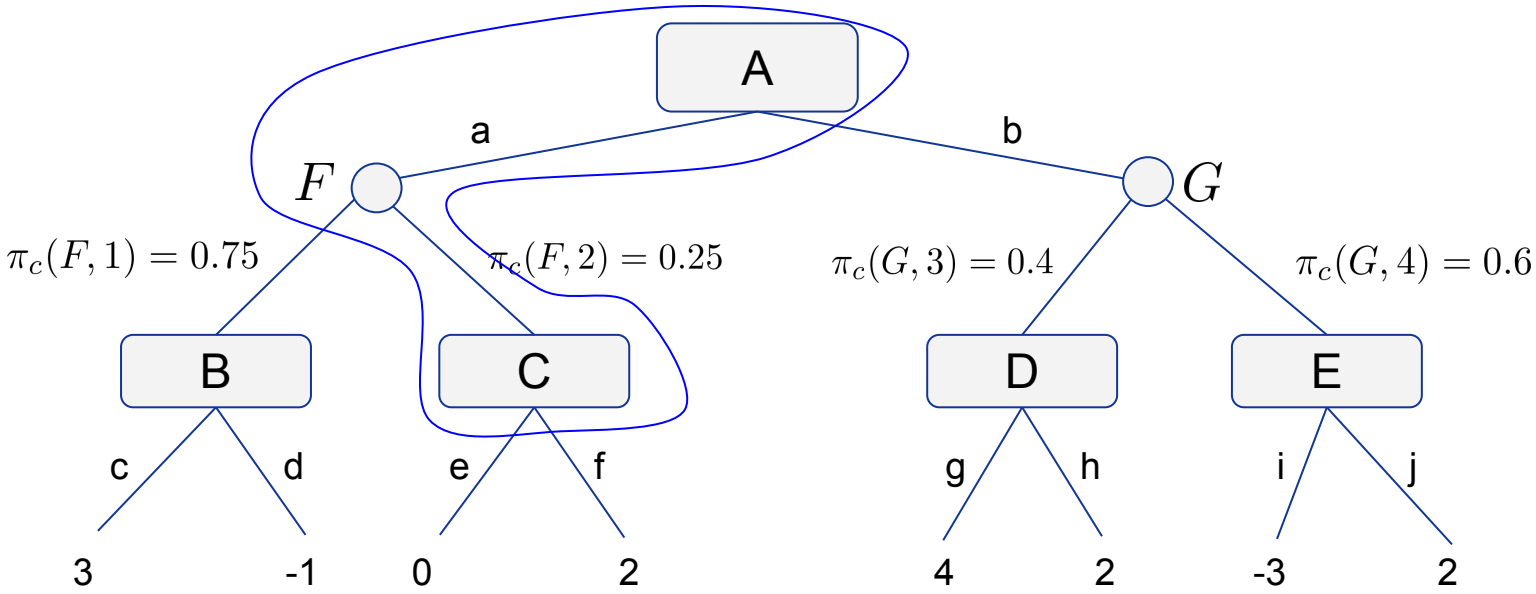
(A, a, F, 1, B, c) is a *terminal* history.

Terminal history A.K.A. Episode



(A, a, F, 1, B, c) is a *terminal* history. (A, b, G, 3, D, g) is a another terminal history.

Prefix (non-terminal) Histories

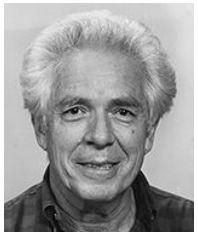


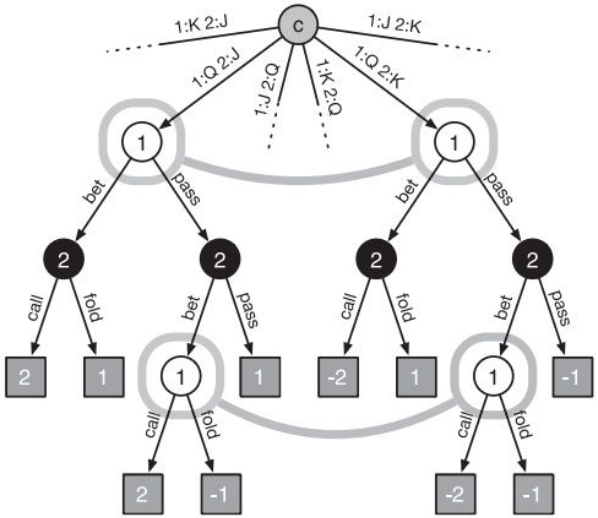
(A, a, F, 2, C) is a history. It is a *prefix* of (A, a, F, 2, C, e) and (A, a, F, 2, C, f).

Partially Observable Zero-Sum Games

Kuhn (simplified) poker

- Players start w/ 2 chips
- Each: ante 1 chip
- 3-card deck
- 2 actions: pass, bet
- Reward: money diff





Terminology

- An **information state,** s, corresponds to a *sequence of observations*
  - with respect to the player to play at s

Ante: 1 chip per player, 

, P1 bets (raise)

Terminology

- An **information state,** s, corresponds to a *sequence of observations*
  - with respect to the player to play at s

private observation

Ante: 1 chip per player,  , P1 bets (raise)

Terminology

- An **information state,** s, corresponds to a *sequence of observations*
  - with respect to the player to play at s

private observation

Ante: 1 chip per player,  , P1 bets (raise)

Environment is in one of many **world states** $h \in s$

Terminology

- An **information state,** s, corresponds to a *sequence of observations*
  - with respect to the player to play at s

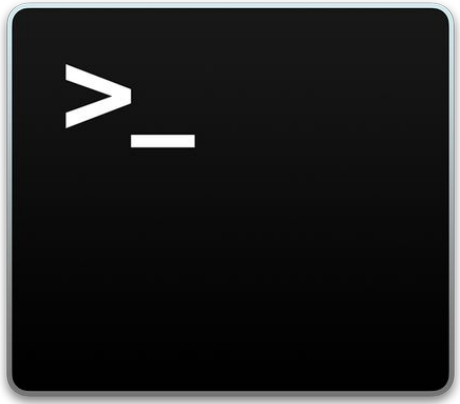
private observation

Ante: 1 chip per player,  , P1 bets (raise)

Environment is in one of many **world states** $h \in s$

full **history** of actions (including nature's!!)

OpenSpiel Live Demo, Part 3



- Imperfect information games, information state strings + vectors
- Feel free to follow along in colab or locally!
- Transcript of demo: `demo3.txt`

Query API

- OpenSpiel designed to be a *generic API* (breadth vs. depth)

Query API

- OpenSpiel designed to be a *generic API* (breadth vs. depth)
- However, sometimes domain-specific knowledge is required.

Query API

- OpenSpiel designed to be a *generic API* (breadth vs. depth)
- However, sometimes domain-specific knowledge is required.

OpenSpiel provides a query API to get knowledge about states:

- `query.h`, `query.cc`
- `python/pybind11/pyspiel.cc`

Query API

- OpenSpiel designed to be a *generic API* (breadth vs. depth)
- However, sometimes domain-specific knowledge is required.

OpenSpiel provides a query API to get knowledge about states:

- `query.h`, `query.cc`
- `python/pybind11/pyspiel.cc`

Currently only one game uses this.

Best File References

First example and API references:

- `examples/example.cc`,
- `python/examples/example.py`
- `python/examples/matrix_game_example.py`
- `python/egt/dynamics_test.py`
- `python/examples/kuhn_policy_gradient.py`
- `python/examples/tic_tac_toe_qlearner.py`
- `python/examples/independent_tabular_qlearning.py`

Demo transcripts: `demo1.txt`, `demo2.txt`, `demo3.txt`

Thank You!

OpenSpiel: A Framework for Reinforcement Learning in Games

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, Jonah Ryan-Davis

(Submitted on 26 Aug 2019 (v1), last revised 31 Dec 2019 (this version, v5))

OpenSpiel is a collection of environments and algorithms for research in general reinforcement learning and search/planning in games. OpenSpiel supports n-player (single- and multi- agent) zero-sum, cooperative and general-sum, one-shot and sequential, strictly turn-taking and simultaneous-move, perfect and imperfect information games, as well as traditional multiagent environments such as (partially- and fully- observable) grid worlds and social dilemmas. OpenSpiel also includes tools to analyze learning dynamics and other common evaluation metrics. This document serves both as an overview of the code base and an introduction to the terminology, core concepts, and algorithms across the fields of reinforcement learning, computational game theory, and search.

- Paper: https://arxiv.org/abs/1908.09453
- Github: github.com/deepmind/open_spiel/

DeepMind

The end and thank you

Marc Lanctot
lanctot@google.com
mlanctot.info

10/03/2020