Trabalho 2 - IA

Grupo: Jaine Conceição, Matheus Santos Almeida e Thomé Pereira

Definição do Trabalho

VERSÃO 1

VERSÃO 2

VERSÃO 3

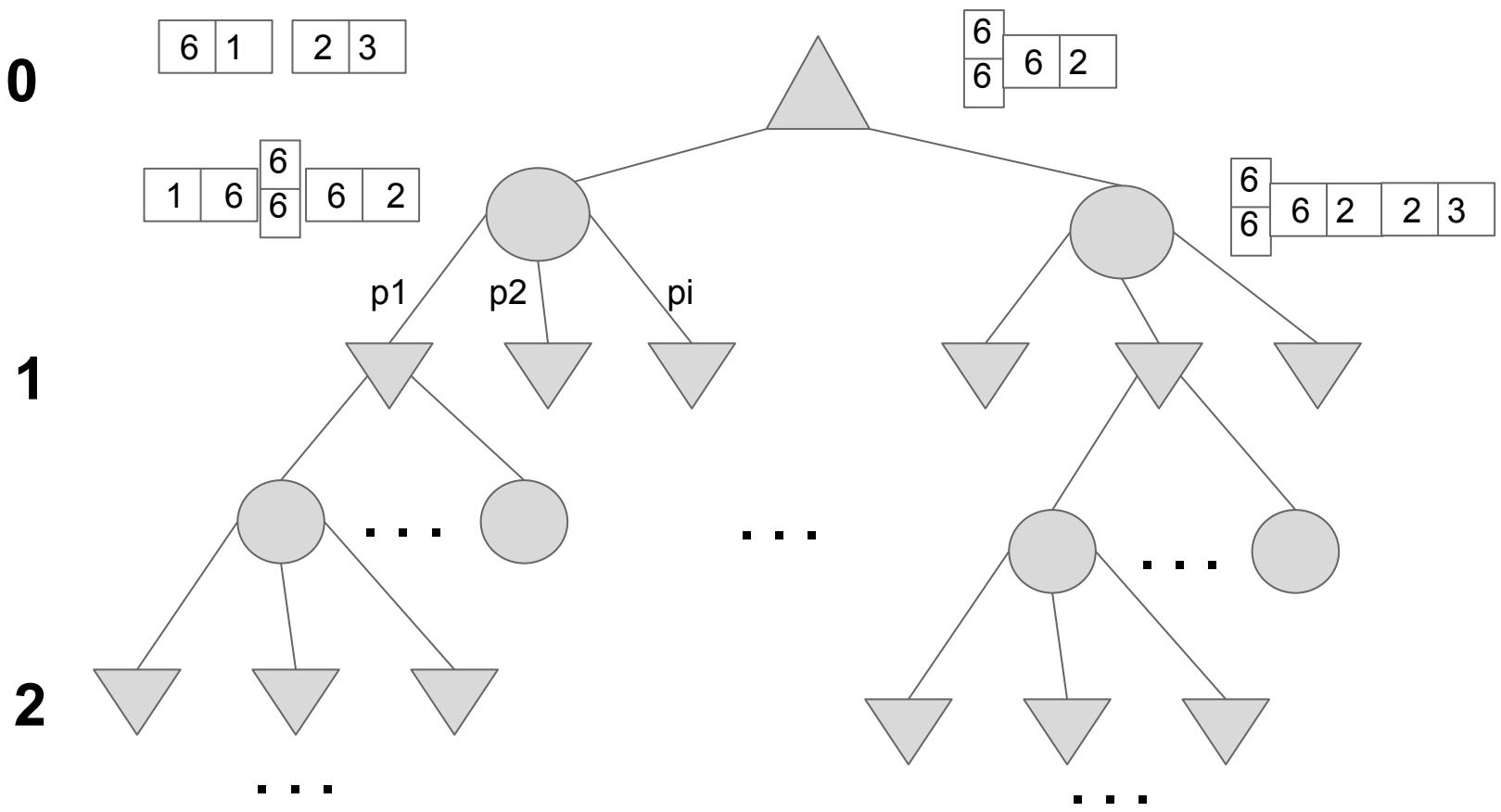
Versão 1:

- Agente utilizando expectiminimax;
- 4 jogadores individuais com 7 pedras cada.

Formalização da versão 1

- **Estados:** GameState(to_move, pedras, pedras_restantes, ponta1, ponta2, moves, utility**)**;
- **Jogadores:** P = {0,1,2,3};
- **Ações:** Lista com pedras e respectivas pontas em que essas pedras se encaixam;
- **Função de transição:** Recebe uma pedra e a ponta em que ela encaixa e o estado atual e retorna um novo estado com essa pedra colocada na mesa;
- **Utilidade:** Se o agente expectiminimax vence, então utilidade é 4, senão utilidade é -1.

Árvore Expectiminimax da versão 1



Implementação 1

Experimentação 1

- **0** - Expectiminimax com eval_0
- **1** - Expectiminimax com eval_1
- **2** - Expectiminimax com eval_2
- **3** - Random

```
def eval_0 (self, state, player):
        n = len(state.pedras)
        m = len(state.pedras_restantes)/3
        buchas = self.buchas(state.pedras)
        soma_pedras = self.soma_pedra(state)
        delta = m - n
        return -0.9*buchas - 0.2*n + 0.7*delta -
0.4*soma_pedras
```

Experimentação 1

- **0** - Expectiminimax com eval_0
- **1** - Expectiminimax com eval_1
- **2** - Expectiminimax com eval_2
- **3** - Random

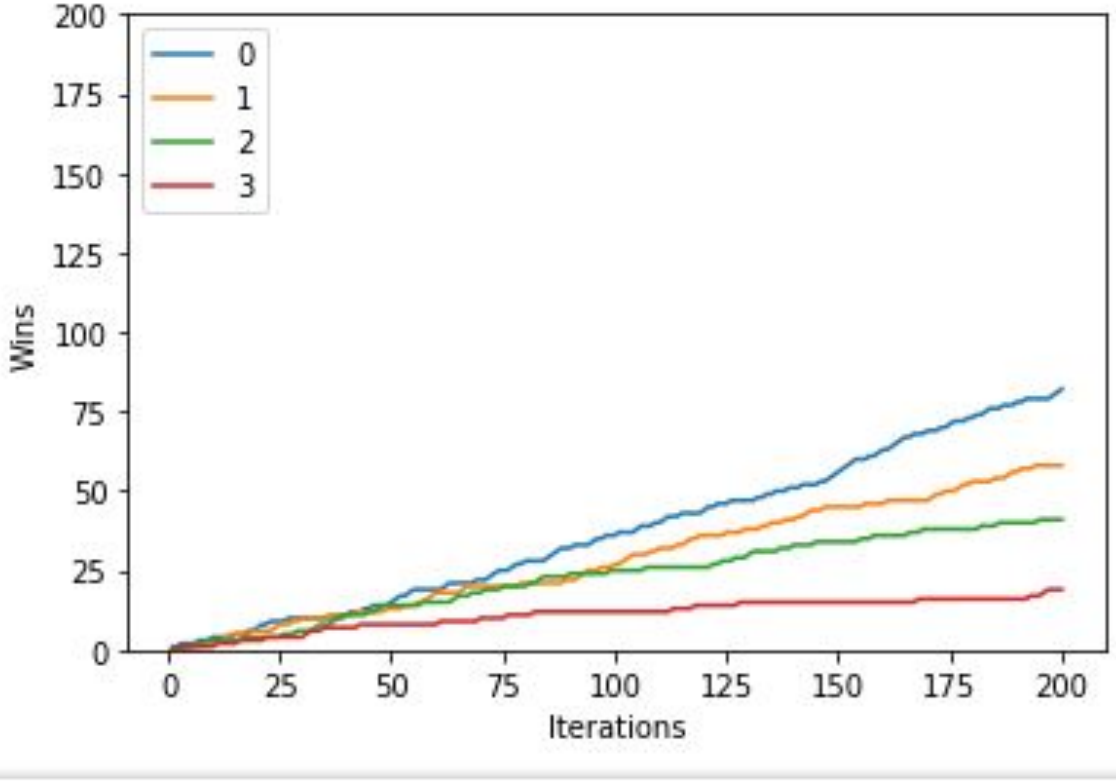
```
def eval_1 (self, state, player):
        movimentos = self.moves(...)
        diversidade = checa_diversidade(...)
        return 0.5*len(movimentos) +
                        0.6*diversidade
```

Experimentação 1

- **0** - Expectiminimax com eval_0
- **1** - Expectiminimax com eval_1
- **2** - Expectiminimax com eval_2
- **3** - Random

```
def eval_2 (self, state, player):
        n = len(state.pedras)
        m = len(state.pedras_restantes)/3
        buchas = self.buchas(state.pedras)
        soma_pedras = self.soma_pedra(state)
        delta = m - n
        movimentos = self.moves(...)
        diversidade = checa_diversidade(...)
        return -0.9*buchas - 0.2*n + 0.7*delta -
        0.4*soma_pedras +0.5*len(movimentos)
        + 0.6*diversidade
```

Experimentação 1



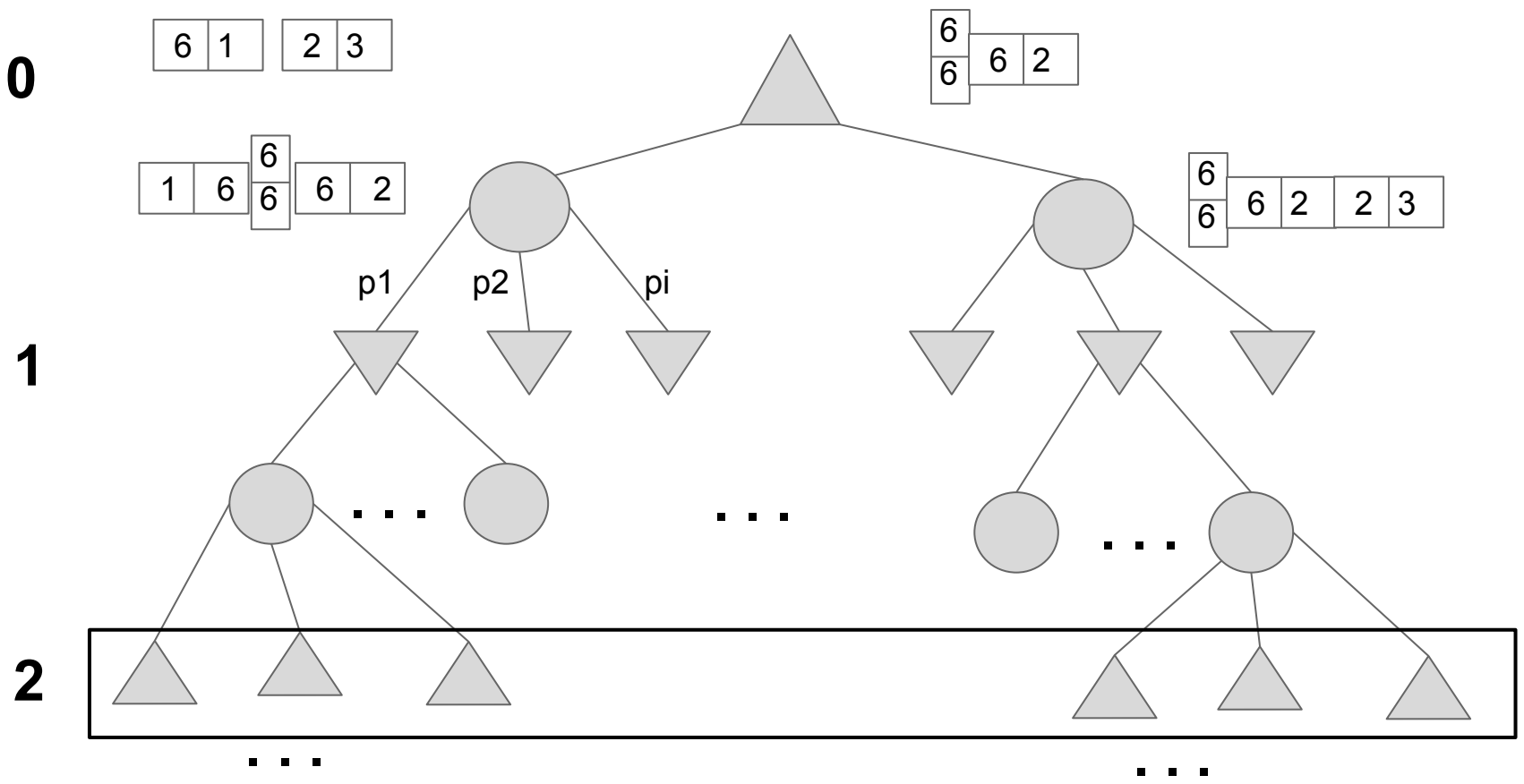
Versão 2:

- Agente utilizando expectiminimax;
- Partidas em duplas com 4 jogadores e 7 pedras cada.

Formalização da versão 2

- **Estados:** GameState(to_move, pedras, pedras_restantes, ponta1, ponta2, moves, utility, **num_pedras=(7,7,7,7)**);
- **Jogadores:** P = {0,1,2,3};
- **Ações:** Lista com pedras e respectivas pontas em que essas pedras se encaixam;
- **Função de transição:** Recebe uma pedra e a ponta em que ela encaixa e o estado atual e retorna um novo estado com essa pedra colocada na mesa;
- **Utilidade:** Se o agente expectiminimax **ou seu parceiro** vence, então utilidade é 4, senão utilidade é -1.

Árvore Expectiminimax da versão 2



Implementação 2

Experimentação 2

- **0 e 2** com eval_amiga
- **1 e 3** com eval_egoista

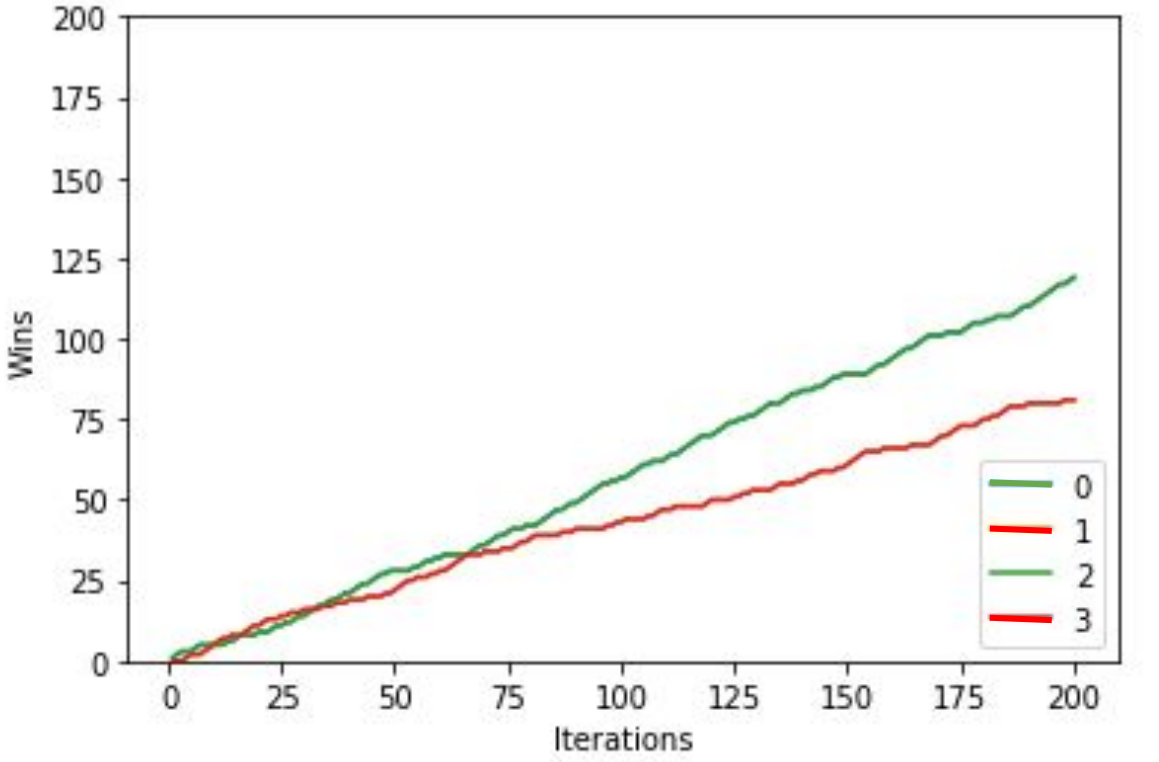
```
def eval_egoista (self, state, player):
        n = len(state.pedras)
        m = len(state.pedras_restantes)/3
        buchas = self.buchas(state.pedras)
        soma_pedras = self.soma_pedra(state)
        delta = m - n
        return -0.9*buchas - 0.2*n + 0.7*delta -
0.4*soma_pedras
```

Experimentação 2

- **0 e 2** com eval_amiga
- **1 e 3** com eval_egoista

```
def eval_amiga (self, state, player):
        n = len(state.pedras)
        m = len(state.pedras_restantes)/3
        p = state.num_pedras[(player+2)%4]
        buchas = self.buchas(state.pedras)
        soma_pedras = self.soma_pedra(state)
        delta = m - (n+p)
        return -0.9*buchas - 0.2*n + 0.7*delta -
0.4*soma_pedras - 0.2*p
```

Experimentação 2



Versão 3:

- Agente utilizando Q-learning;
- 4 jogadores individuais com 7 pedras cada.

Formalização da versão 3

- **Estados:** GameState(pedras, pedras_restantes, pontas);

- **Ações:** Lista com pedras e respectivas pontas em que essas pedras se encaixam;

- **Recompensa:** R(s,a,s') = ???

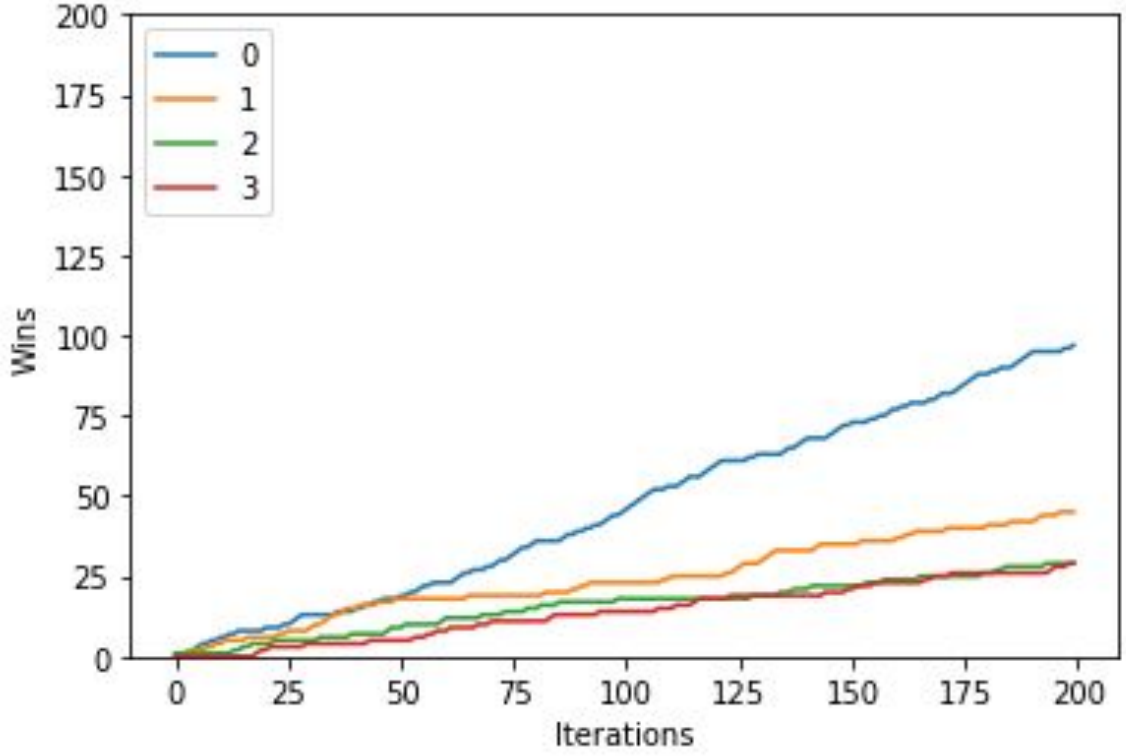
Implementação 3

Experimentação 3

- **0 -** Agente Q-learning
- **1 -** Expectiminimax com eval_0
- **2 -** Random
- **3 -** Random

```
def eval_0 (self, state, player):
        n = len(state.pedras)
        m = len(state.pedras_restantes)/3
        buchas = self.buchas(state.pedras)
        soma_pedras = self.soma_pedra(state)
        delta = m - n
        return -0.9*buchas - 0.2*n + 0.7*delta -
0.4*soma_pedras
```

Experimentação 3



Obrigado!

Probabilidade

- **Ponta 1:**
  numero_pedras_ponta1 / numero_pedras_restantes

- **Ponta 2:**
  numero_pedras_ponta2 / numero_pedras_restantes

- **Pingar/passar:**
  1 - (numero_movimentos / numero_pedras_restantes)

Função de Avaliação da versão 1

- **Levou em consideração as seguintes características:**
  - **N** = Número de pedras do agente
  - **M** = Número de pedras restantes / 3
  - **Buchas** = Número de buchas na mão do agente
  - **Movimentos** = Número de movimentos possíveis do agente naquele estado
  - **Soma de pedras** = Soma dos pontos das pedras na mão do agente
  - **Diversidade** = Número de naipes diferentes na mão do agente

Probabilidade

- **Ponta 1:**
  numero_pedras_ponta1 / numero_pedras_restantes

- **Ponta 2:**
  numero_pedras_ponta2 / numero_pedras_restantes

- **Pingar/passar:**
  1 - (numero_movimentos / numero_pedras_restantes)

Funções de Avaliação da versão 2

- **N** = Número de pedras do agente
- **M** = Número de pedras restantes / 3
- **Buchas** = Número de buchas na mão do agente
- **Soma de pedras** = Soma dos pontos das pedras na mão do agente
- **p** = Número de pedras do parceiro
- **Delta** = m - (n+p)

```
def eval_0 (self, state, player):
        n = len(state.pedras)
        m = len(state.pedras_restantes)/3
        buchas = self.buchas(state.pedras)
        soma_pedras = self.soma_pedra(state)
        delta = m - n
        return -0.9*buchas - 0.2*n + 0.7*delta -
0.4*soma_pedras
```