

PLAYER 1

HIGHSCORE 2500

HEARTS

PLAYER 2

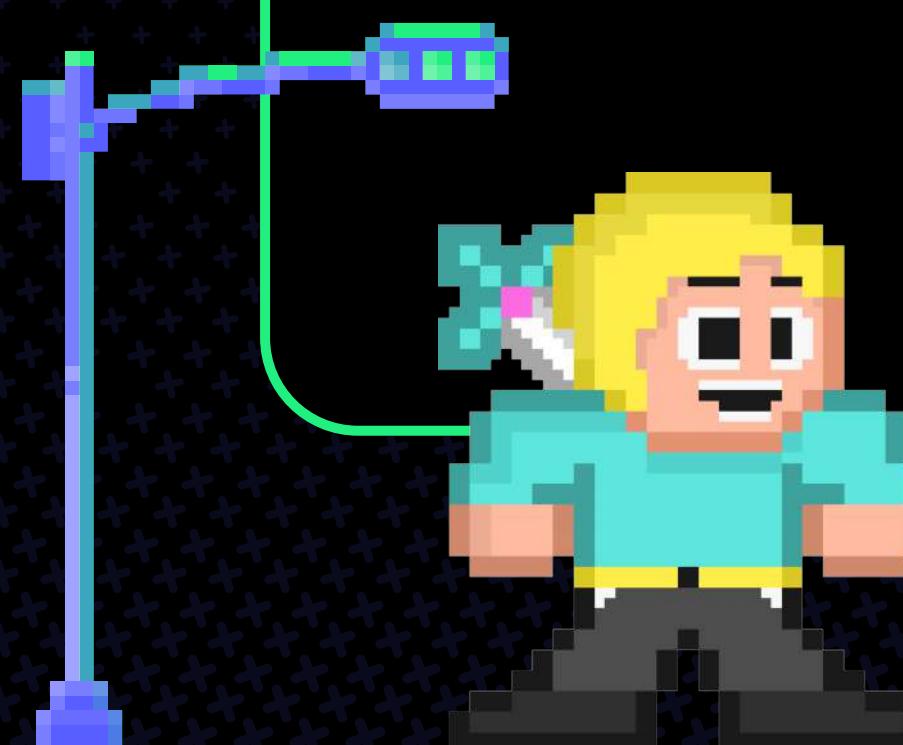
AUTÔMATOS
UTILIZADOS EM
NPC's

START

MENU

SIGN IN

ALUNOS: EMANUEL LOPES E
STENIO FONSECA



MENU

01

07

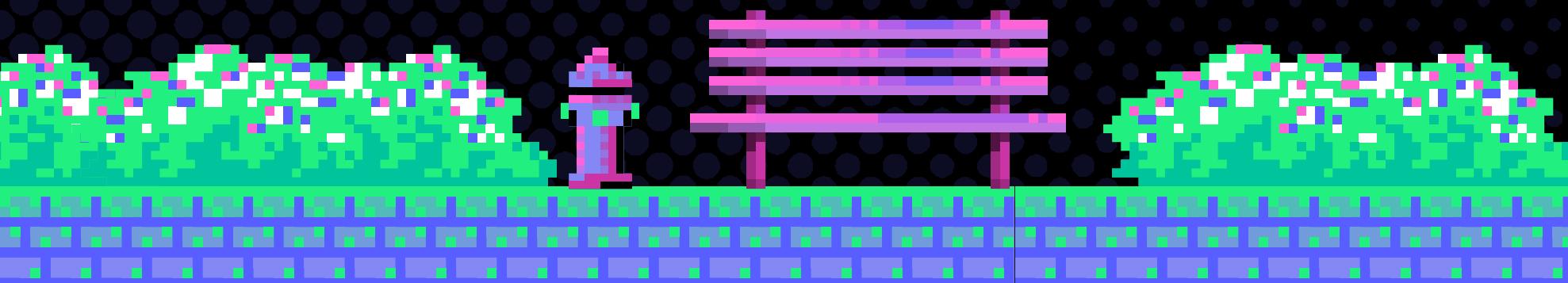
12



SUMÁRIO

◆ TÓPICOS DE APRESENTAÇÃO

- ◆ INTRODUÇÃO
- ◆ FUNDAMENTAÇÃO TEÓRICA
- ◆ JOGO 1: ARKANOID
- ◆ JOGO 2: INTERAÇÃO COM NPC
- ◆ ADENDO : NPC AFD COMPLEXO
- ◆ LIMITAÇÕES
- ◆ CONCLUSÃO



MENU

01

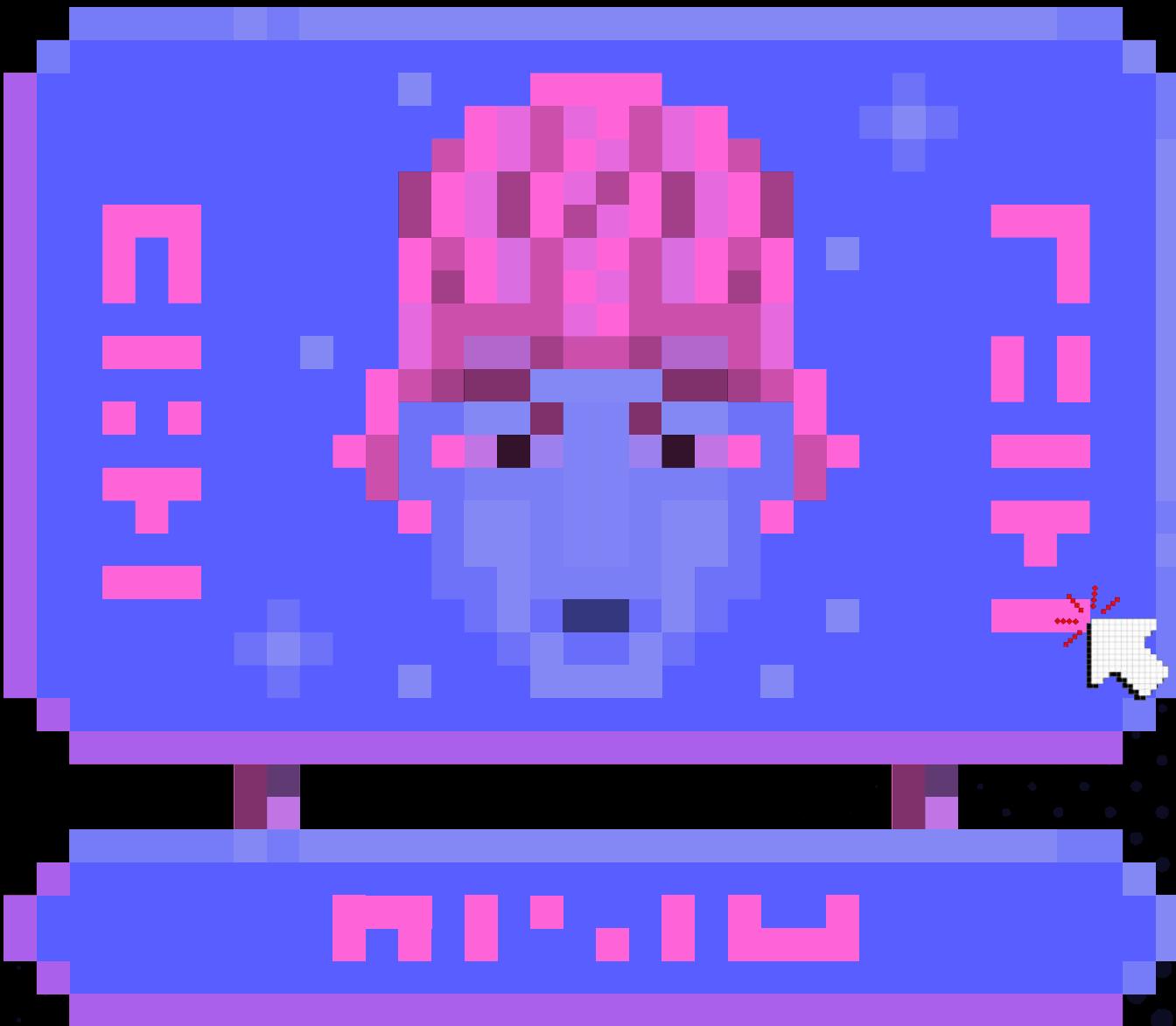
07

12



INTRODUÇÃO

INTRODUÇÃO SOBRE OS PROJETOS
DESENVOLVIDOS



→ 01 ⚡ 07 ★ 12



INTRODUÇÃO

- ➡ CRIAÇÃO DE UM JOGO ARKANOID FUNCIONAL EM PYTHON COM PYXEL
- ➡ IMPLEMENTAÇÃO DE NPCS INTERATIVOS USANDO AUTÔMATOS AFD/AFN/AP
- ➡ APLICAÇÃO PRÁTICA DE TEORIA DA COMPUTAÇÃO (AFD, AFN, AP) EM JOGOS INTERATIVOS

SIGN IN



*FUNDAMENTAÇÃO TEÓRICA



AFO

Máquina de estados finitos em que, para cada estado e símbolo de entrada, existe uma única transição possível

AFO

$$M = (Q, \Sigma, \delta, q_0, F)$$

- Q é um conjunto finito de estados;
- Σ é um alfabeto finito de símbolos de entrada;
- $\delta : Q \times \Sigma \rightarrow Q$ é a função de transição, que associa de forma determinística cada par (estado, símbolo) a um único próximo estado;
- $q_0 \in Q$ é o estado inicial;
- $F \subseteq Q$ é o conjunto de estados finais ou de aceitação.

AFD

- AFDs reconhecem linguagens regulares, que podem ser expressas por expressões regulares
- São fechadas sob operações como união, interseção, complemento e estrela de Kleene
- Podem ser representadas por gramáticas regulares (tipo 3 na hierarquia de Chomsky)
- Para qualquer AFD, existe um AFD minimizado equivalente, com o menor número possível de estados

AFN

Máquina de estados finitos onde, para um mesmo estado e símbolo de entrada, podem existir múltiplas transições possíveis

AFN

$$M = (Q, \Sigma, \delta, q_0, F)$$

Onde:

$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$, que significa que a função de transição retorna um conjunto de estados

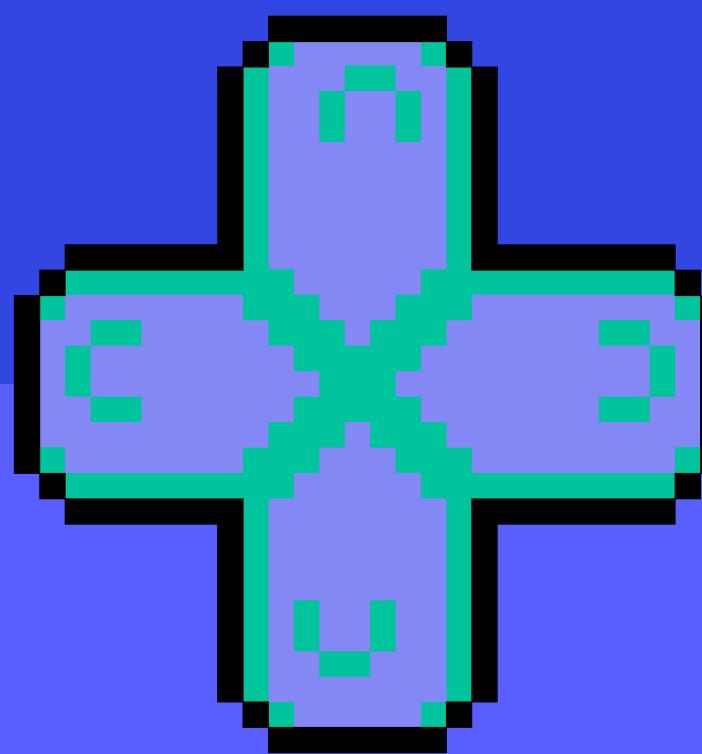
- Permite explorar múltiplos caminhos
- Pode incluir transições- ϵ , ou seja, mudanças de estado sem consumir entrada.

AP

Máquina de estados finitos com uma estrutura adicional de pilha, permitindo o armazenamento e recuperação de símbolos

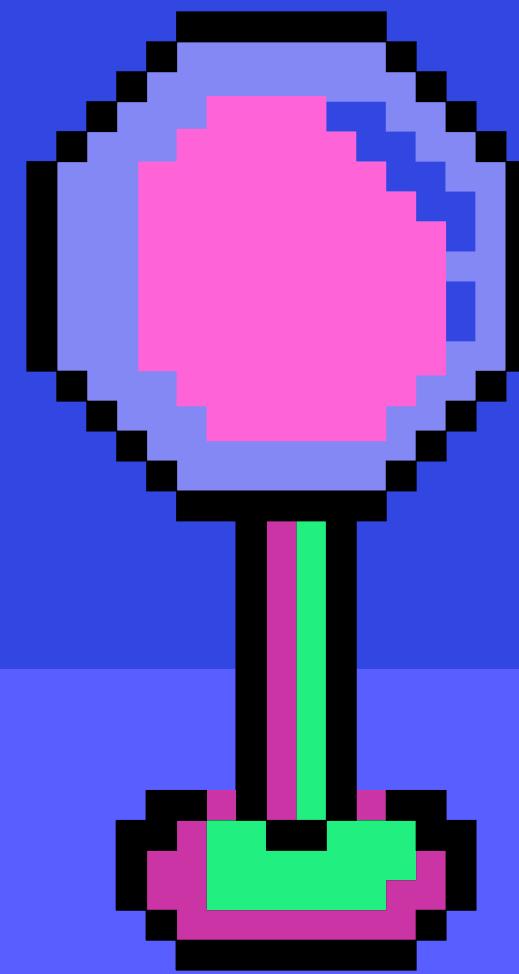
$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

- Q : conjunto finito de estados
- Σ : alfabeto de entrada
- Γ : alfabeto da pilha
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$
é a função de transição com pilha
- $q_0 \in Q$: estado inicial
- $Z_0 \in \Gamma$: símbolo inicial da pilha
- $F \subseteq Q$: conjunto de estados finais



JOGO 1:
ARKANOID

START



OBJETIVOS

- ★ Desenvolver uma implementação interativa do jogo Arkanoid em Python, utilizando o framework Pyxel
- ★ Aplicar conceitos de Autômatos Finitos (AFD e AFN) para controlar o comportamento do jogo, como o movimento da bola e do jogador, além do gerenciamento dos estados do jogo
- ★ Desenvolver os elementos gráficos do jogo (tela, paddle, bola, blocos) de forma simplificada, utilizando Pyxel
- ★ Proporcionar uma compreensão prática dos conceitos teóricos de AFD e AFN, aplicados em um contexto lúdico e visual

MENU



ARQUIVOS .PY

- automato.py
- ball.py
- block.py
- config.py
- game.py
- hud.py
- paddle.py
- visualizador_diagrama.py

ARQUIVOS

- ★ config.py: Define constantes globais do jogo: tamanhos de tela, cores, configurações da bola, blocos e paddle
- ★ game.py: Lógica principal do jogo: inicializa o Pyxel, cria objetos (paddle, bola, blocos), gerencia atualizações, colisões e desenha a tela
- ★ ball.py: Define a classe Ball: movimento da bola, colisões com paddle e blocos, perda de vida (quando a bola cai), desenho da bola
- ★ block.py: Define a classe Block: tipo de bloco (normal, resistente, powerup), quantidade de vidas, gerenciamento de impactos e desenho dos blocos na tela

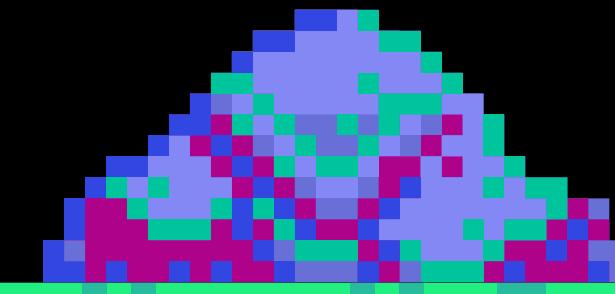
ARQUIVOS

- ★ paddle.py: Define a classe Paddle: paddle controlado pelo jogador, movimentação, limites de tela e desenho
- ★ hud.py: Gera a interface gráfica (HUD): exibe informações como pontuação, vidas restantes e status do jogo
- ★ visualizador_diagrama.py: Gera o diagrama do AFD/AFN do jogo (exemplo: paddle ou bola), usando Graphviz para visualizar os estados e transições
- ★ automato.py: Implementa o AFD/AFN do jogo Arkanoid: comportamento dos elementos como paddle e bola, simulação de estados e transições (ex: "esperando lançamento", "movendo", "batendo")

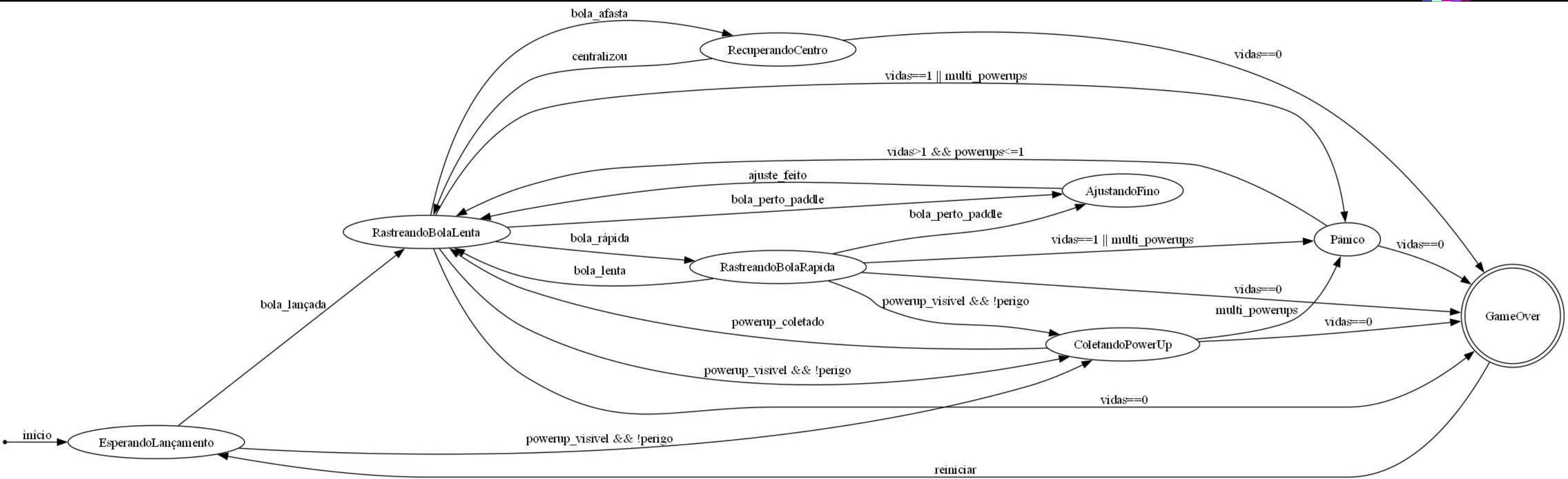
DIAGRAMAS

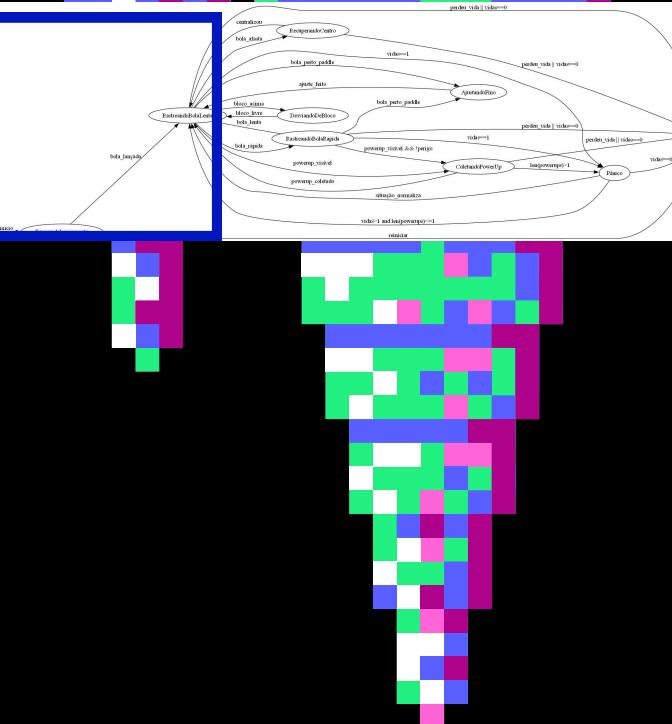
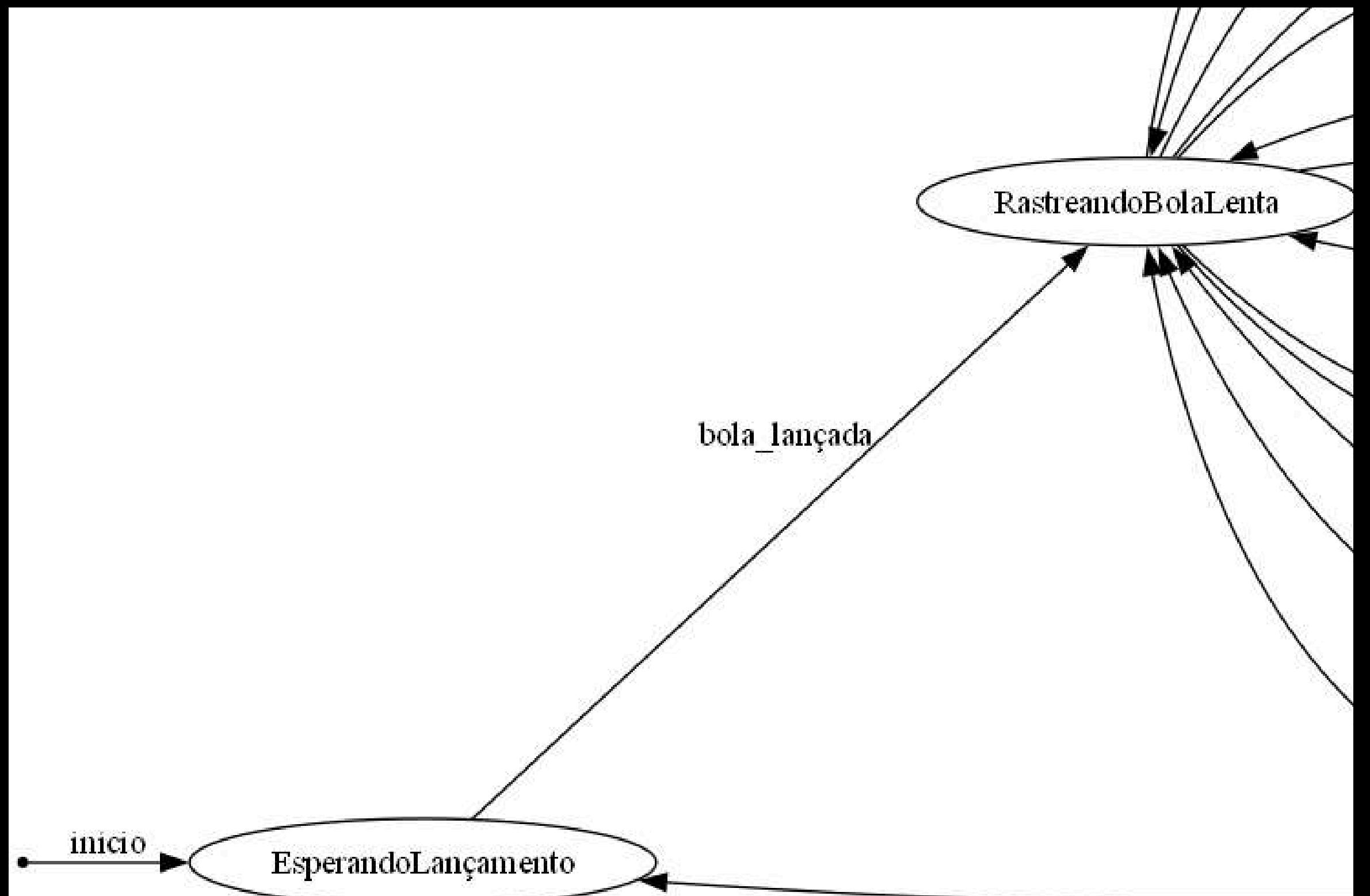


Representação Gráfica dos Estados e Transições

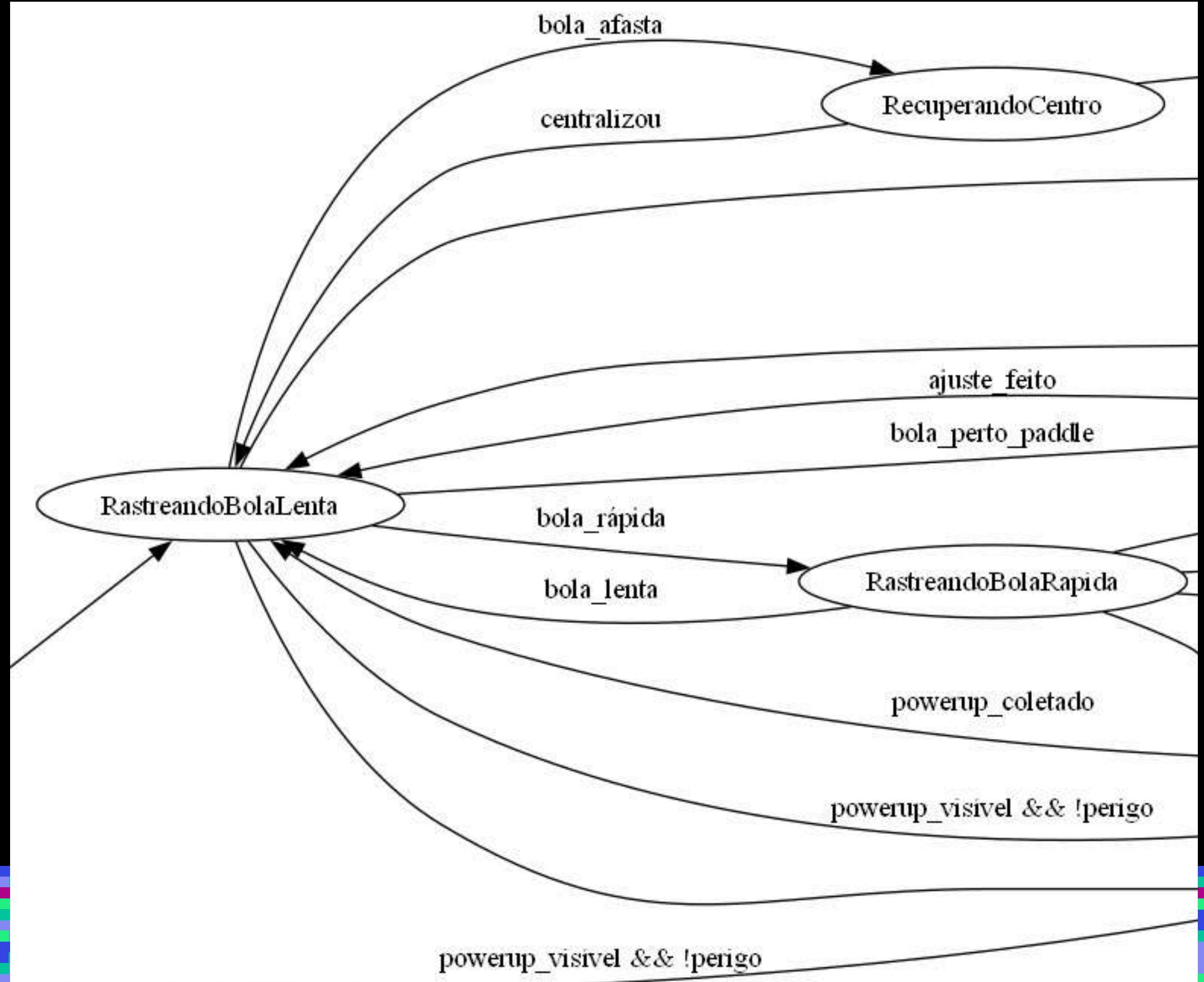


AFO

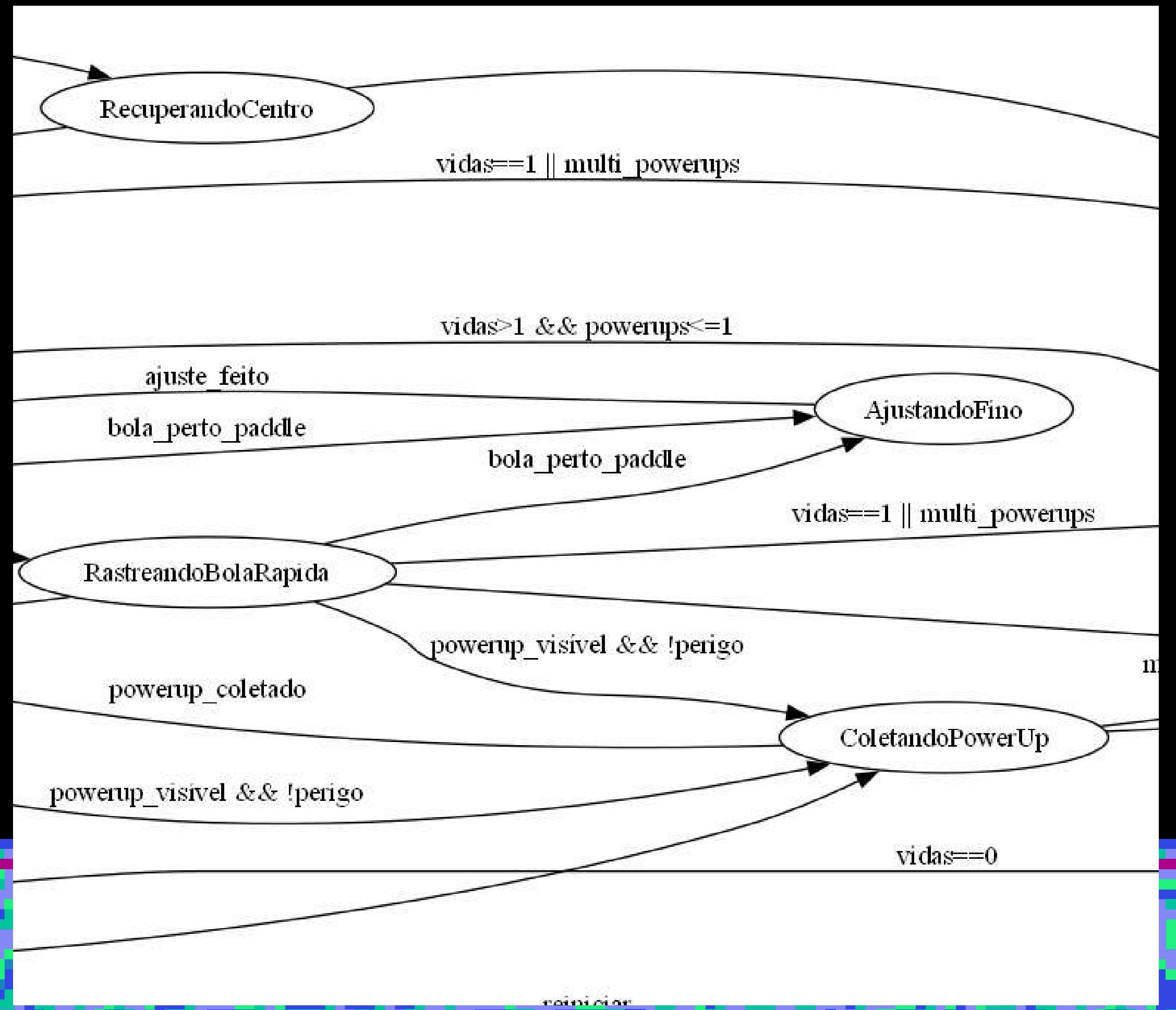




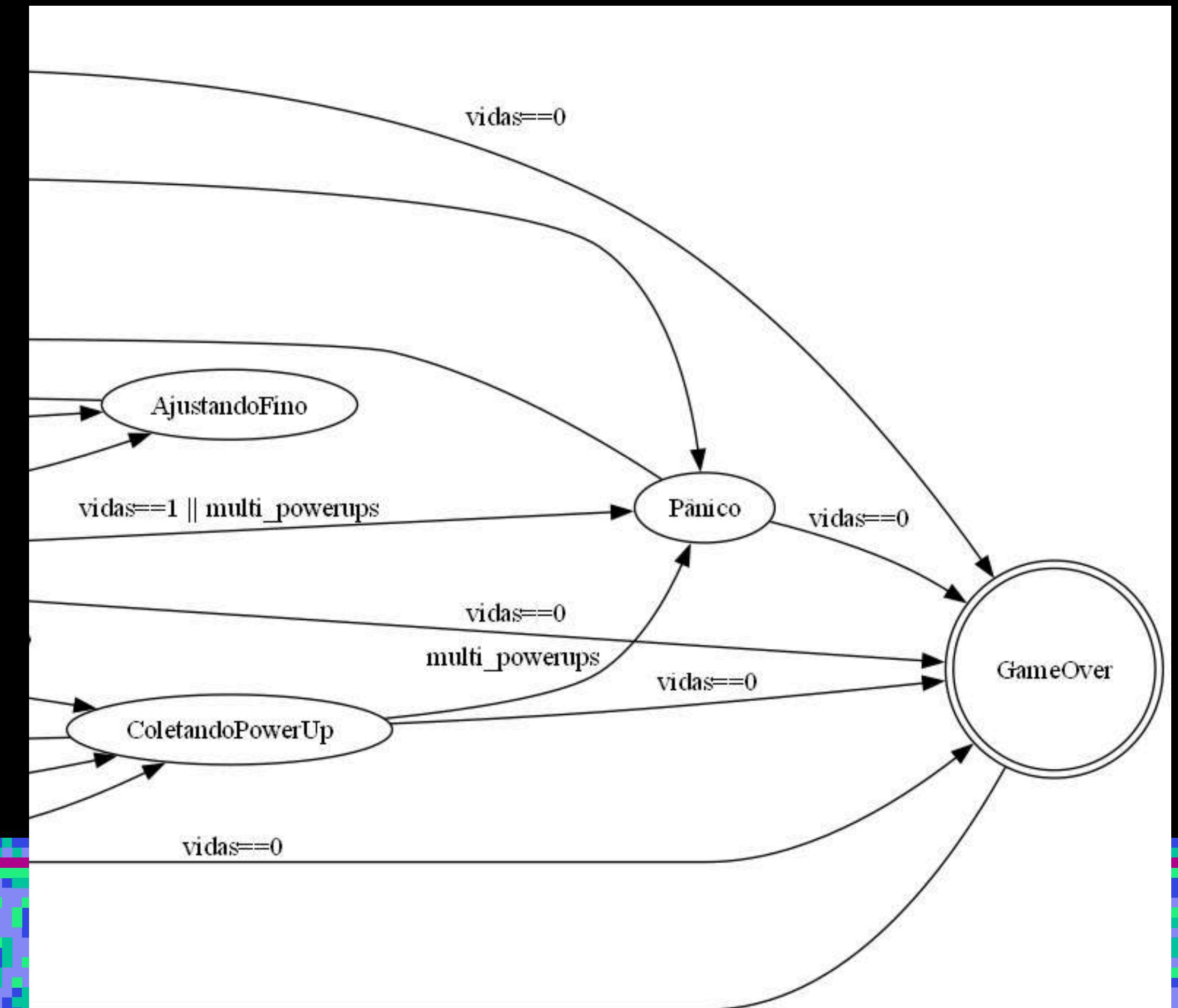
AFD



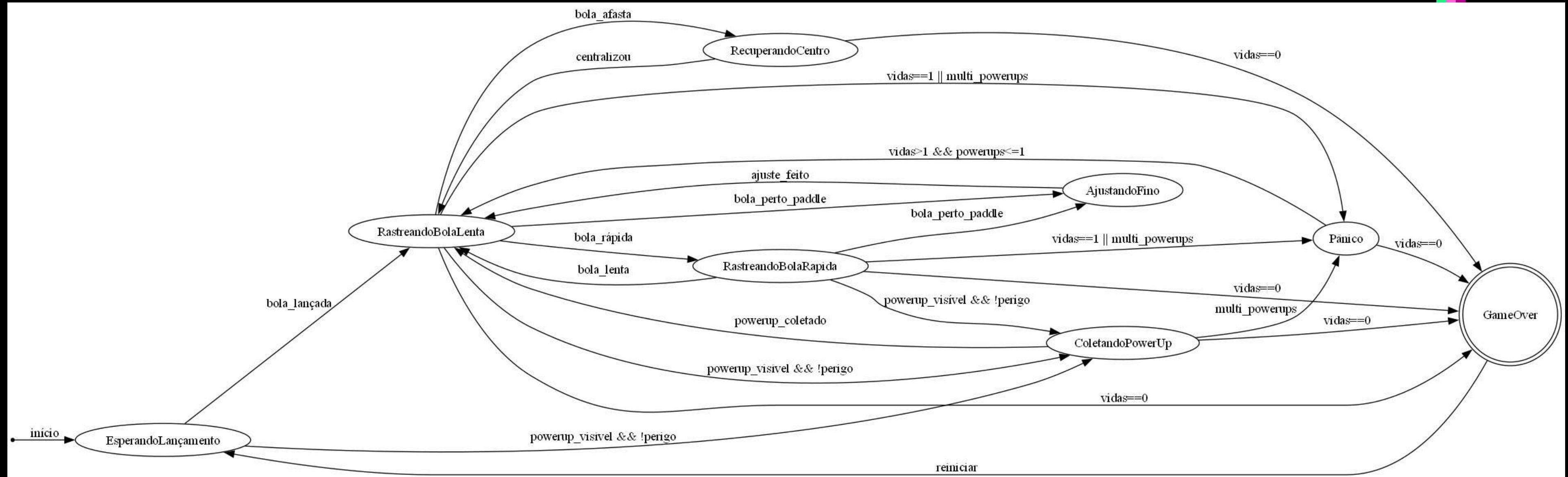
AFO



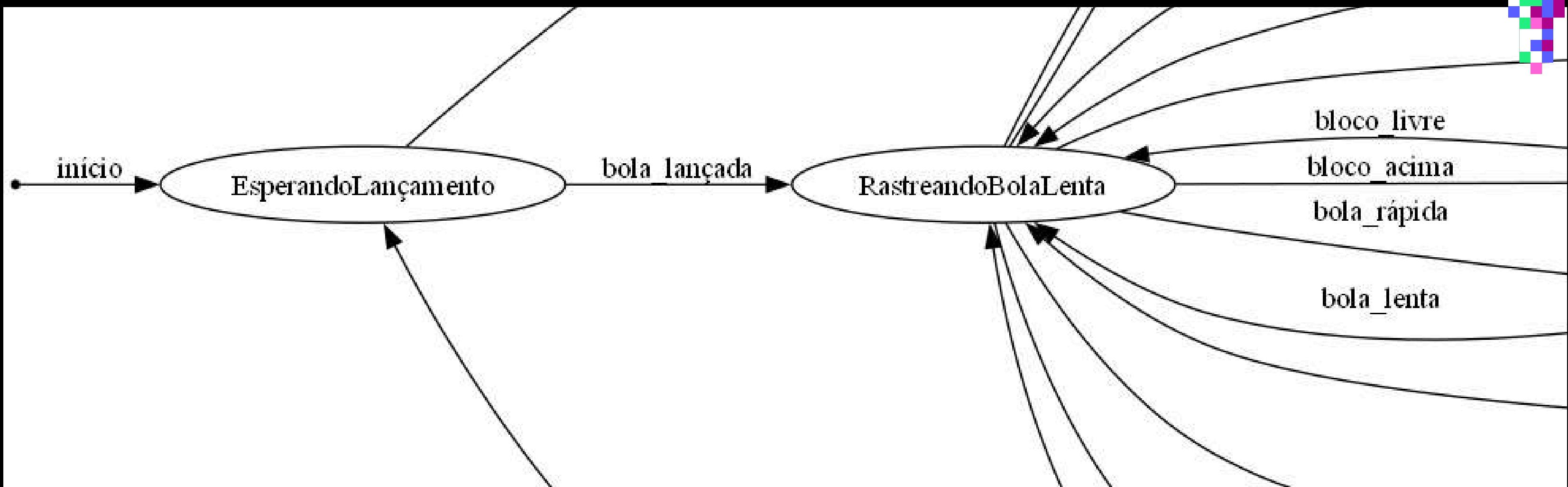
AFO



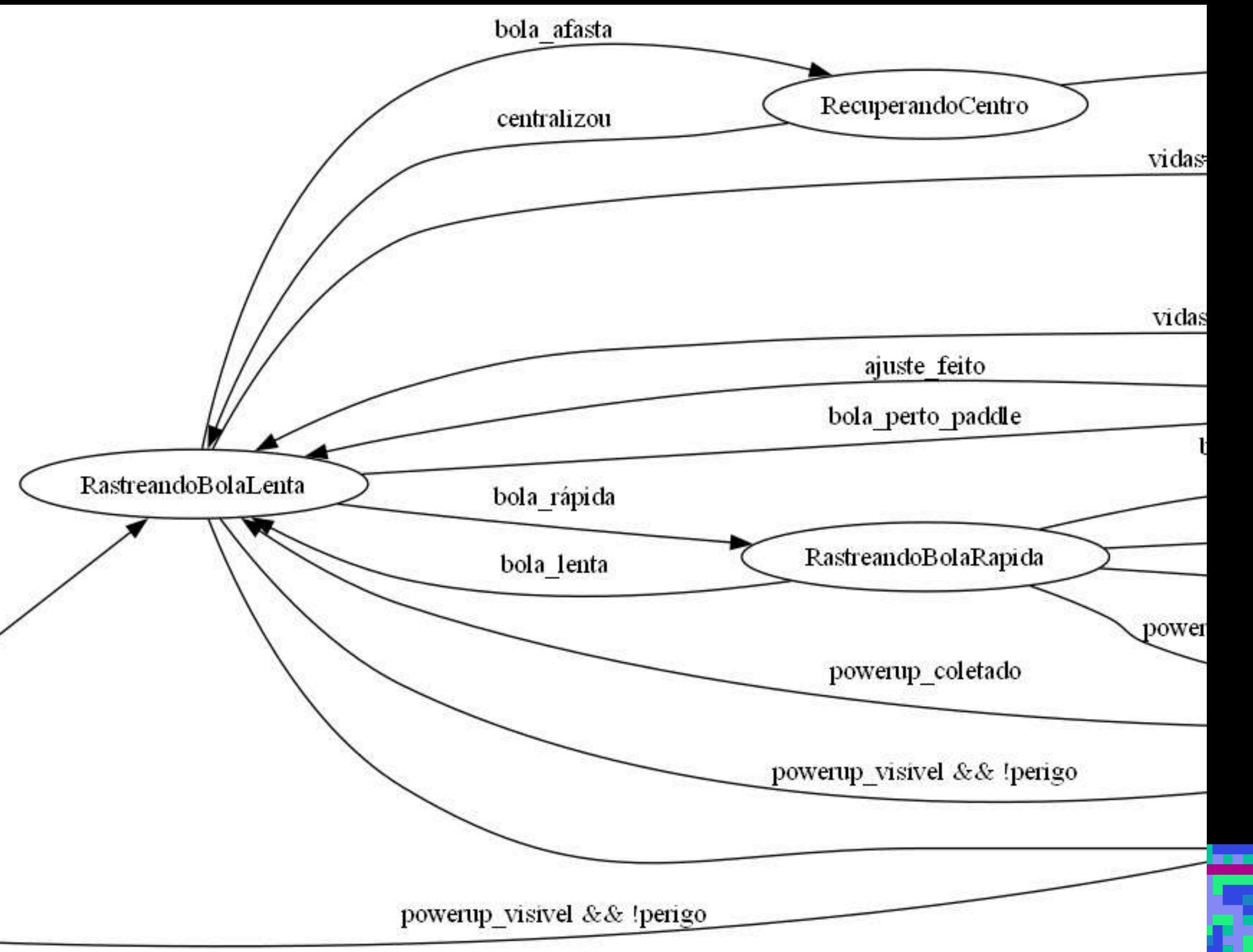
AFN



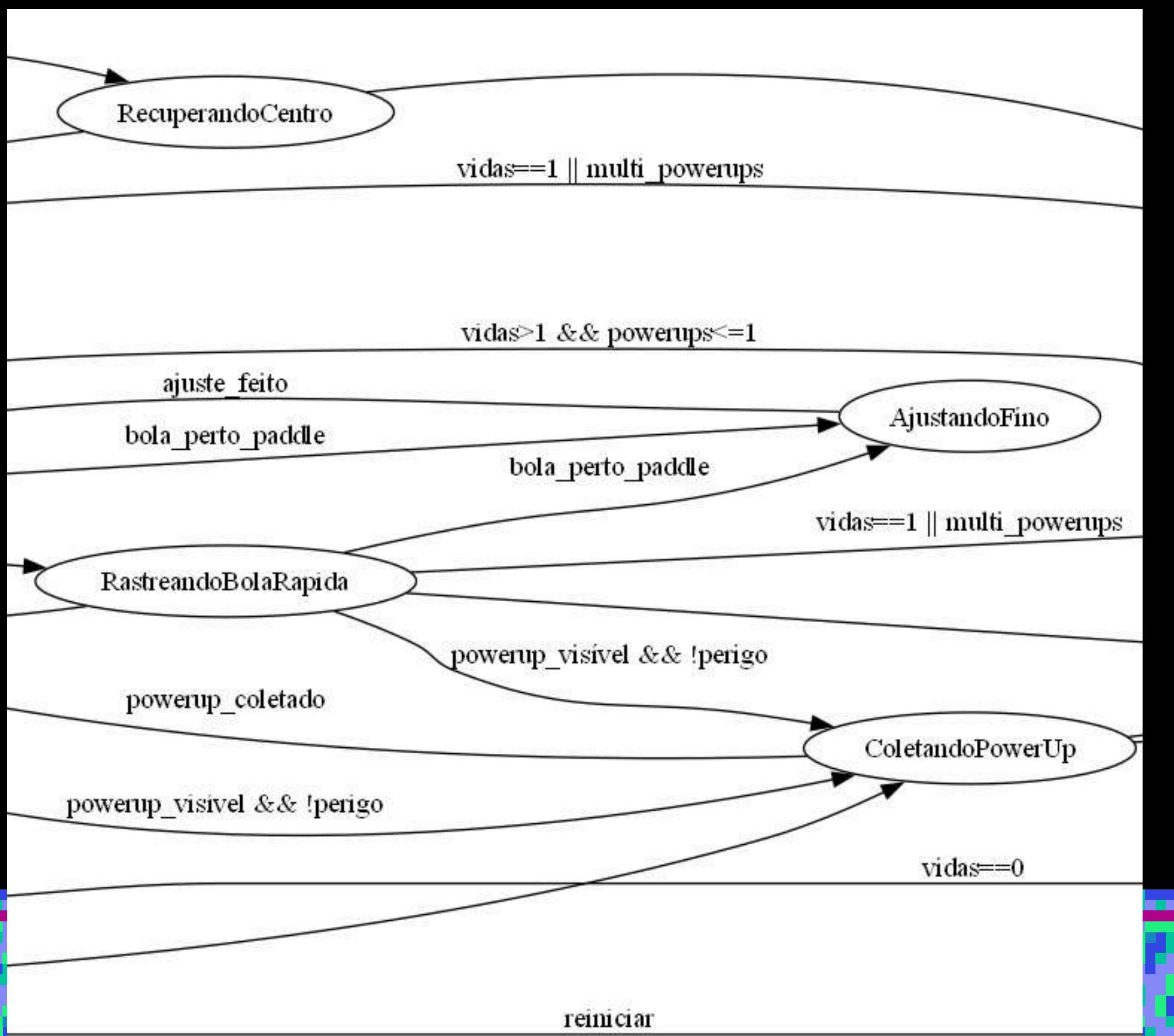
AFN



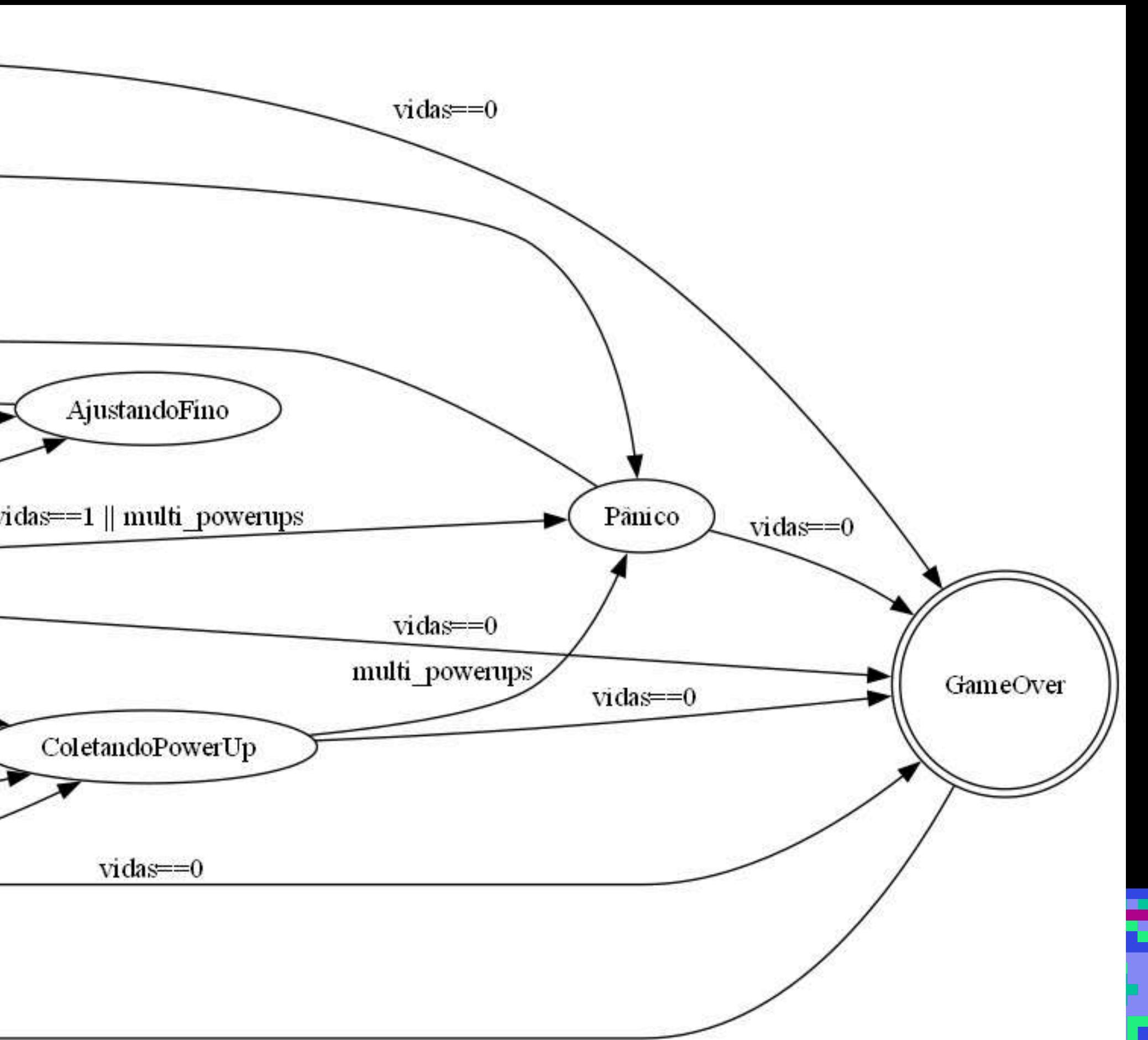
AFN



AFN



AFN

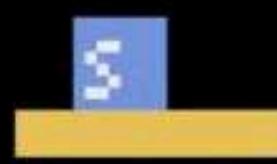
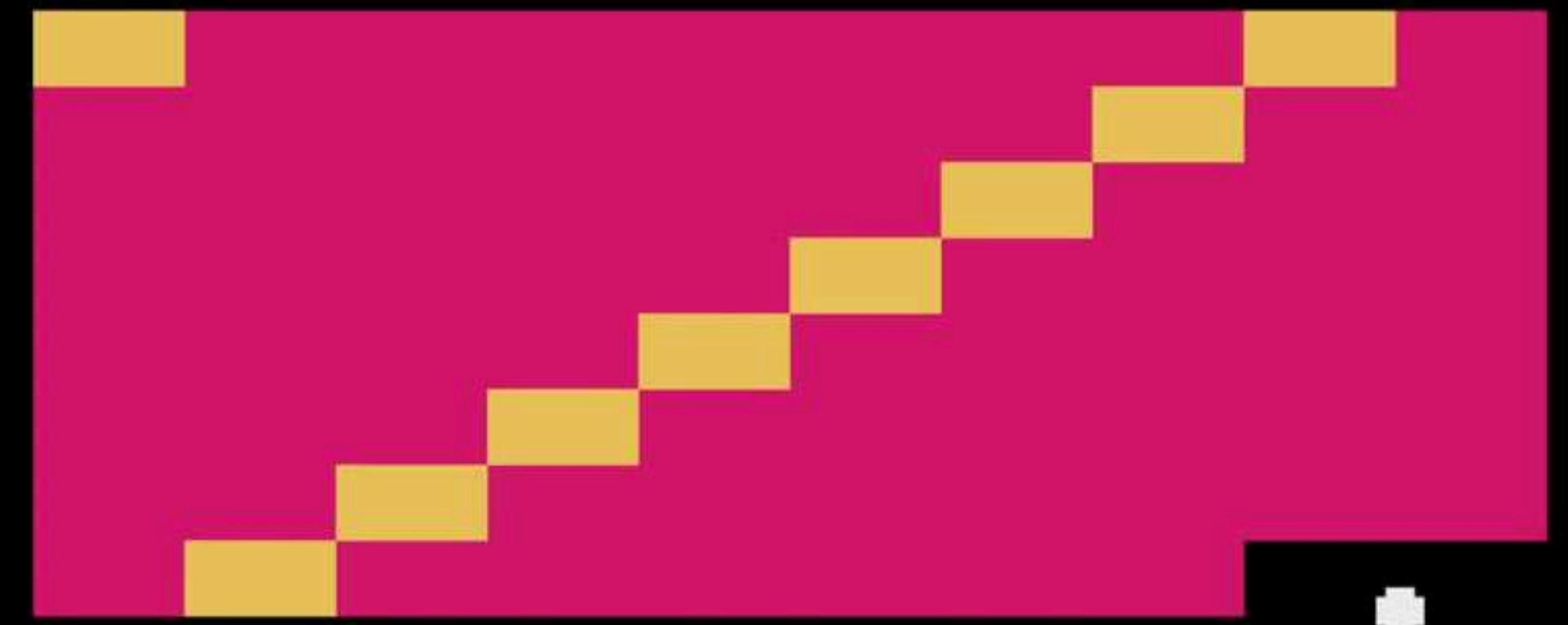


◆ VÍDEO DO PROGRAMA

SCORE: 000020

VIDAS:

0 0 0

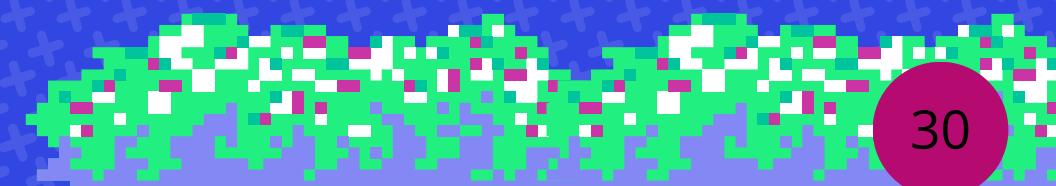


Modo: IA (Tab para alternar)

SIGN IN



LOGO 2:
INTERACAO.COM
NPC



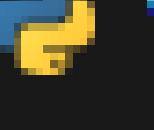
OBJETIVOS

- ★ Implementação de NPCs interativos usando autômatos AFD/AFN/AP
- ★ Transformar conceitos de Teoria da Computação (como AFD, AFN e AP) em um jogo interativo
- ★ No Mercador (AFD) e Ferreiro (Pseudo-AFN com probabilidades), simular decisões como negociar preços, aceitar trocas ou falhar na forja, variando comportamentos com base em inputs e condições específicas
- ★ Proporcionar uma compreensão prática dos conceitos teóricos de AFD, AFN, e AP, aplicados em um contexto lúdico e visual

MENU



ARQUIVOS .PY

-  game.py
-  Imagens.py
-  NpcFerreiro.py
-  NpcInformante.py
-  NpcMercante.py
-  Visualizador.py
-  visualizadordosprite.py
-  VisualizadorGeral.py

ARQUIVOS

- ★ game.py: inicializa o Pyxel, carrega os NPCs (Mercador, Ferreiro, Informante), trata interações, exibe diálogos, e desenha os retratos e a interface
- ★ Imagens.py: Gerencia as sprites e retratos dos NPCs no formato .pyxres. Fornece as coordenadas (u, v) para desenhar cada imagem no jogo
- ★ NpcMercante.py: Implementa o AFD do Mercador: um autômato determinístico para gerenciar as interações de compra e negociação no jogo. Inclui lógica de "verificar compra" com chance de sucesso ou recusa
- ★ NpcFerreiro.py: Define o Pseudo-AFN do Ferreiro: múltiplos caminhos possíveis (forjar, melhorar arma, desafio), permitindo uma narrativa mais complexa

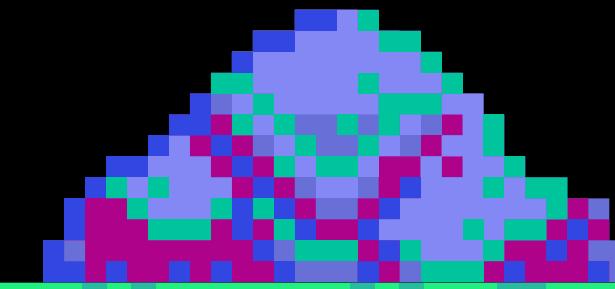
ARQUIVOS

- ★ `NpcInformante.py`: Implementa o AP (Autômato de Pilha) do Informante: permite armazenar perguntas na pilha e exibir explicações personalizadas para cada tema perguntado. Ao final, exibe um resumo do que o jogador aprendeu
- ★ `Visualizador.py`: Cria o diagrama visual dos autômatos individuais (AFD, AFN, AP) dos NPCs, usando Graphviz para gerar imagens representando os estados e transições
- ★ `VisualizadorGeral.py`: Similar ao `Visualizador.py`, mas adaptado para processar múltiplos NPCs e autômatos de diferentes tipos
- ★ `visualizadordosprite.py`: Exibe os sprites carregados no Pyxel para verificar sua posição e tamanho antes de usá-los no jogo. Ajuda a acertar a coordenada dos retratos e animações dos NPCs

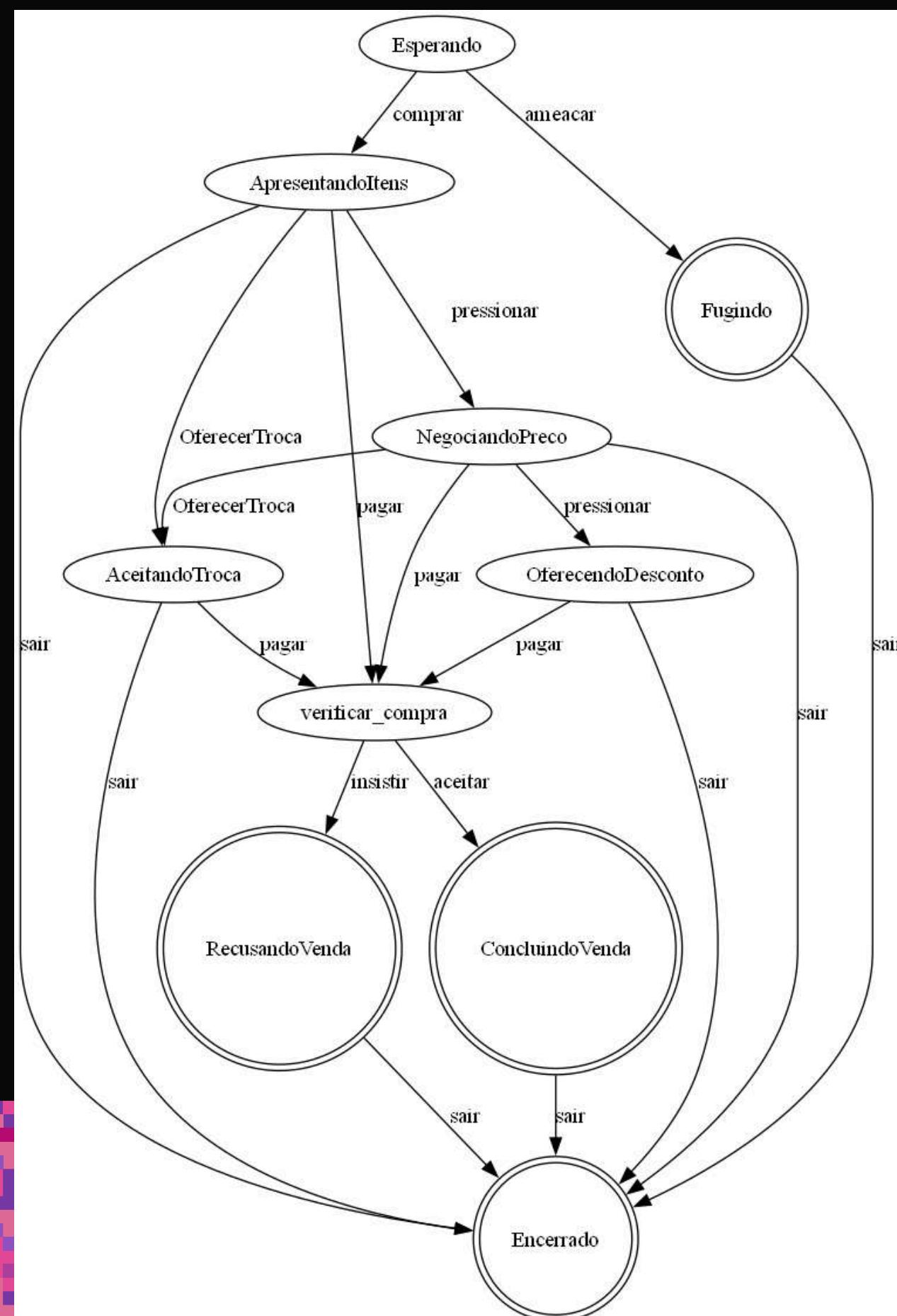
DIAGRAMAS



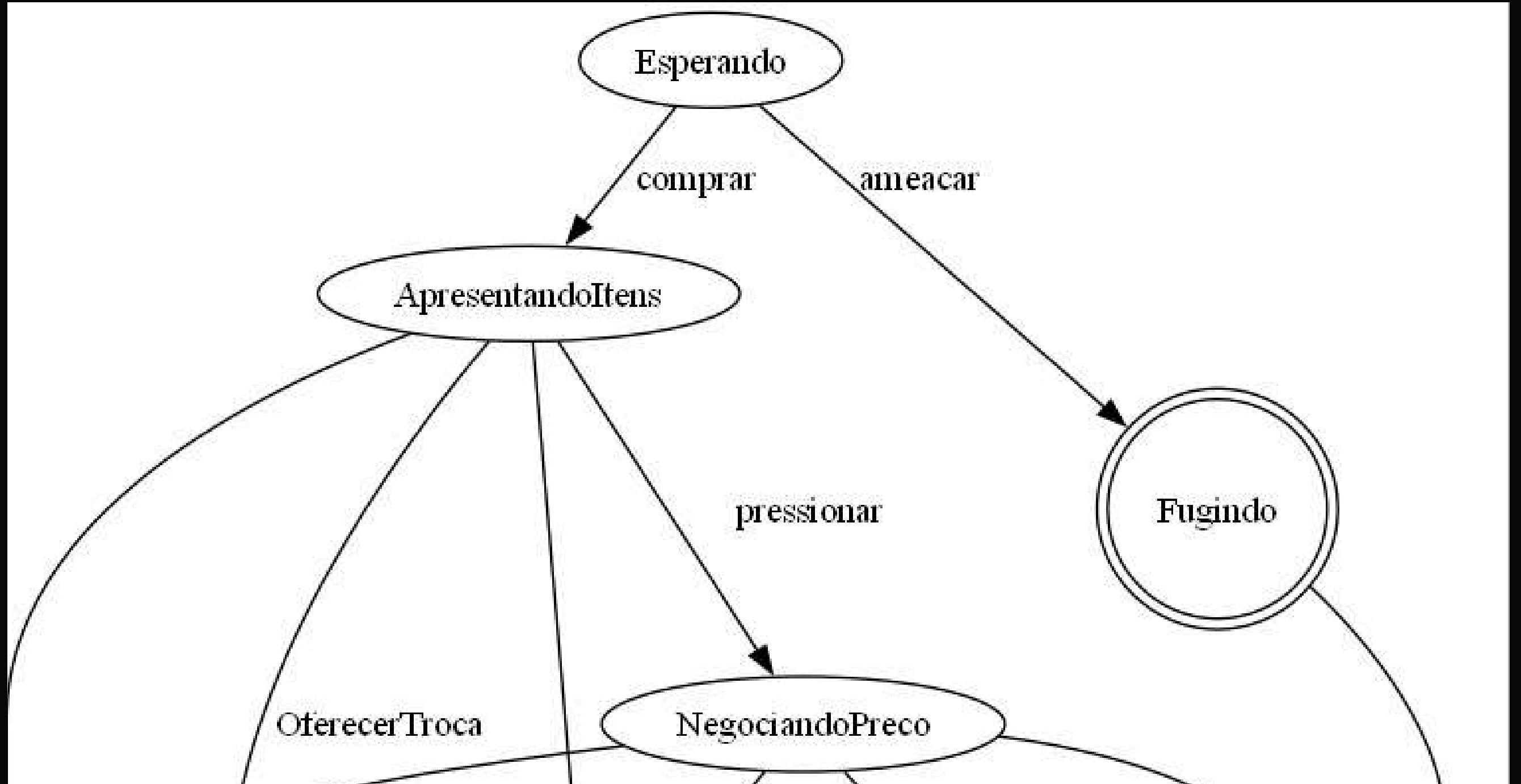
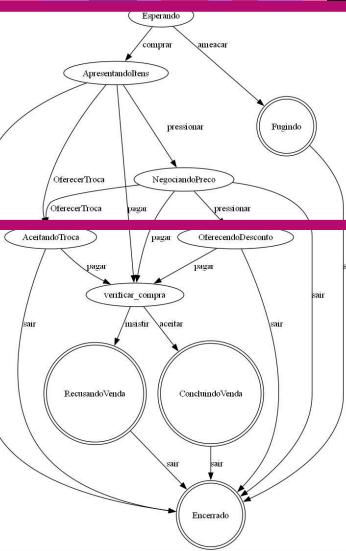
Representação Gráfica dos Estados e Transições



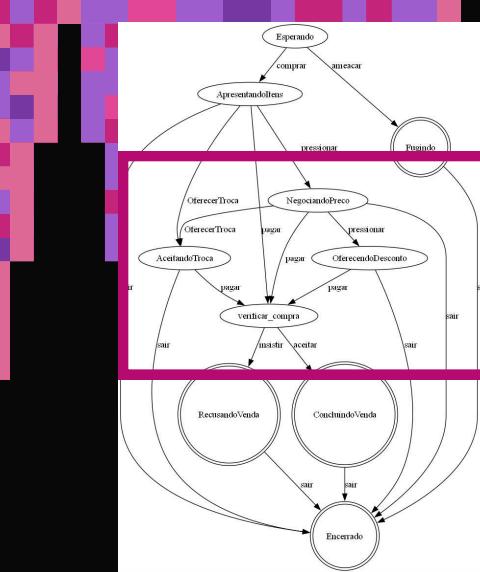
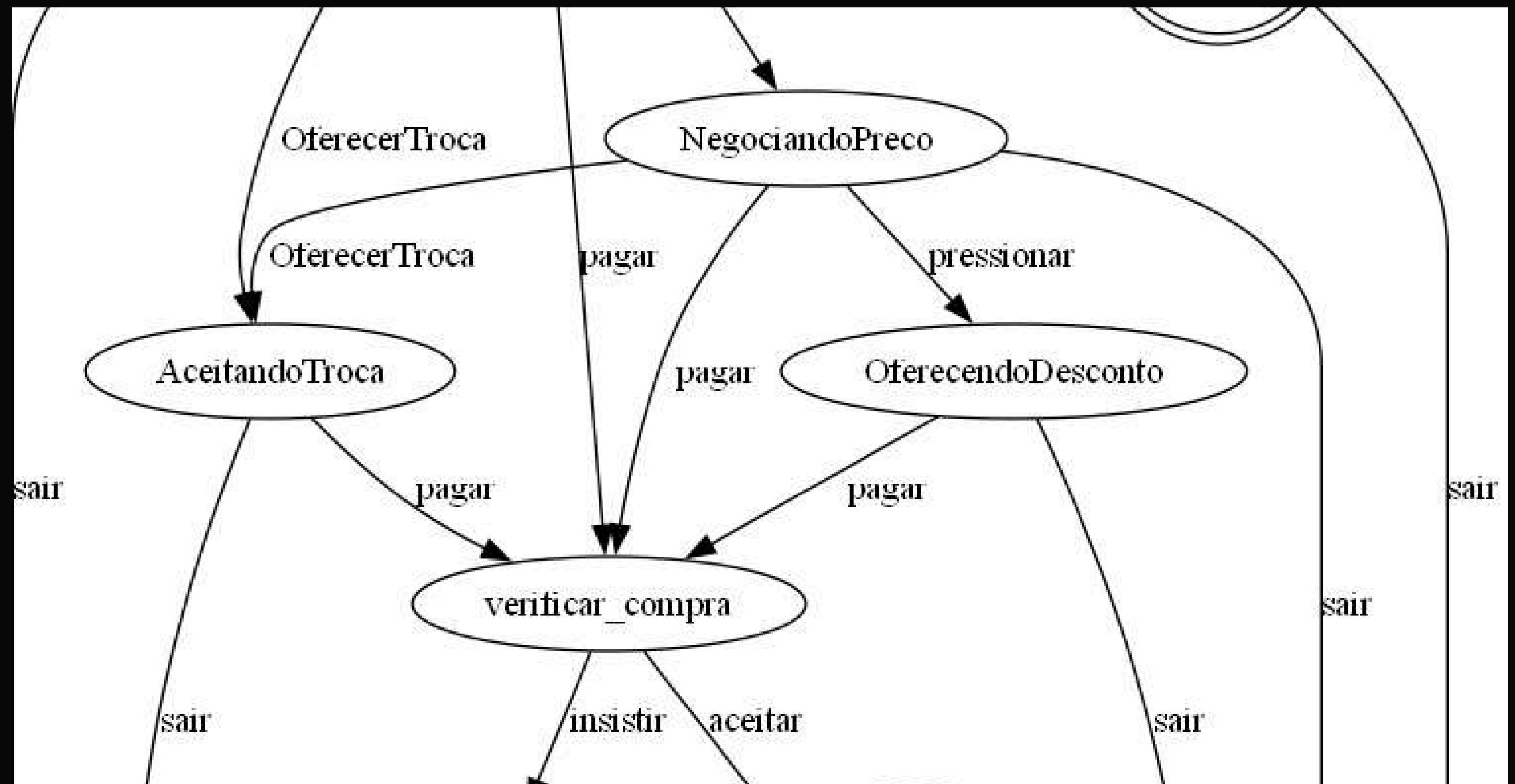
NPC MERCANTE- AFD



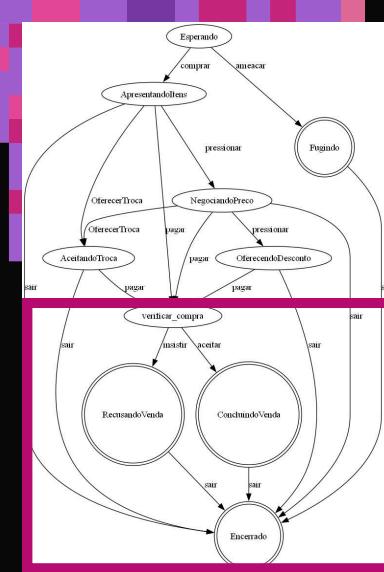
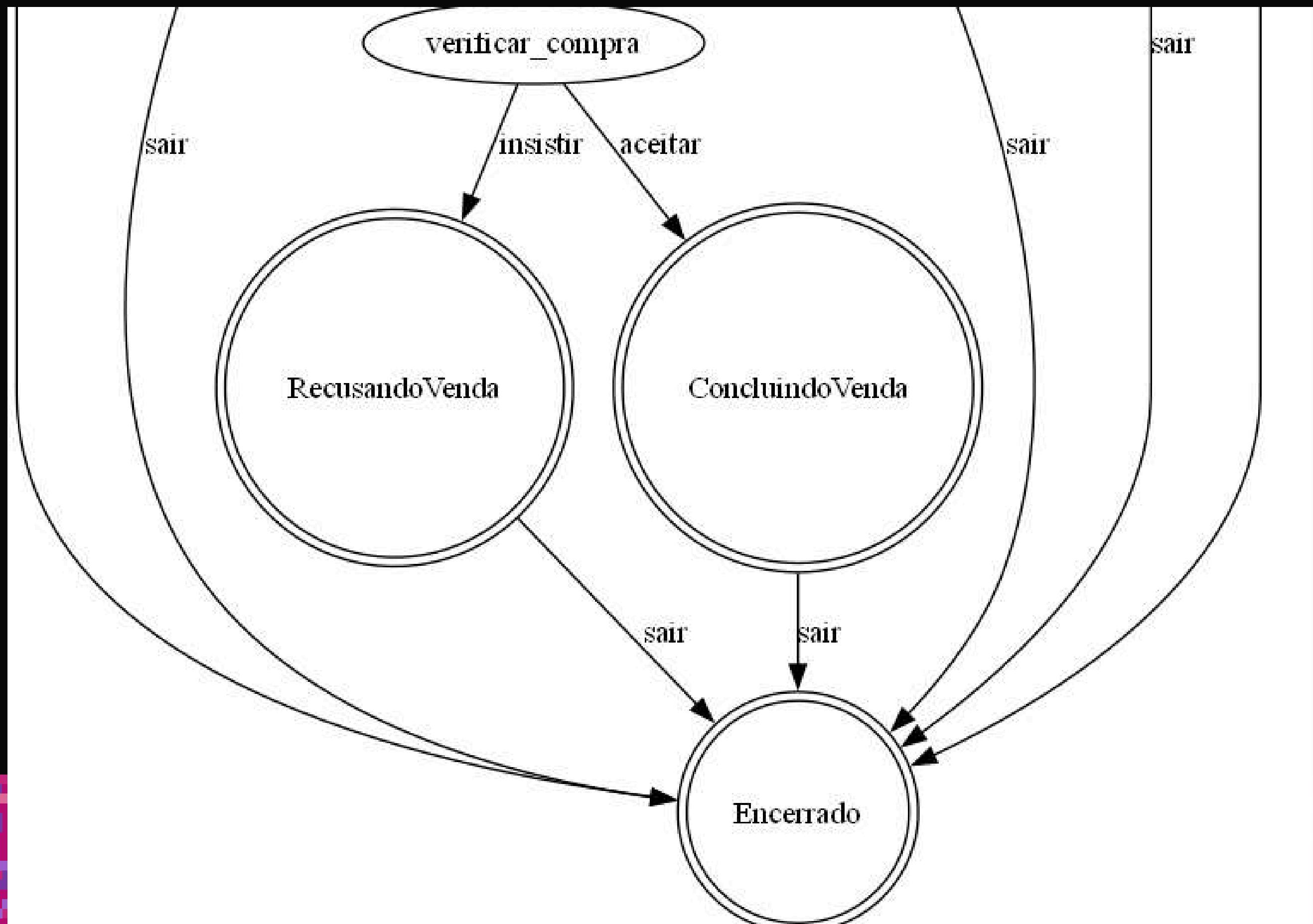
NPC MERCANTE- AFD



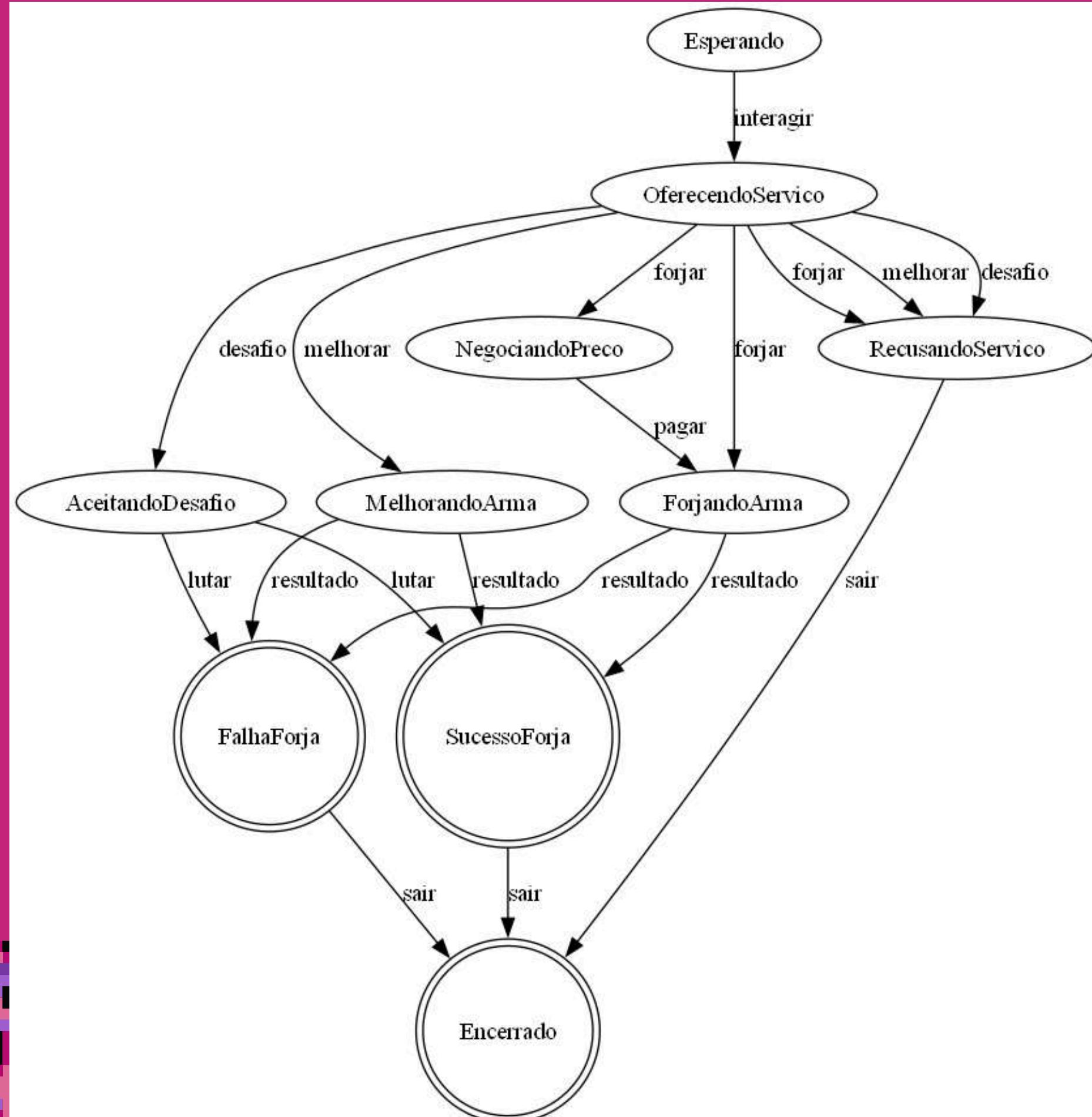
NPC MERCANTE- AFD



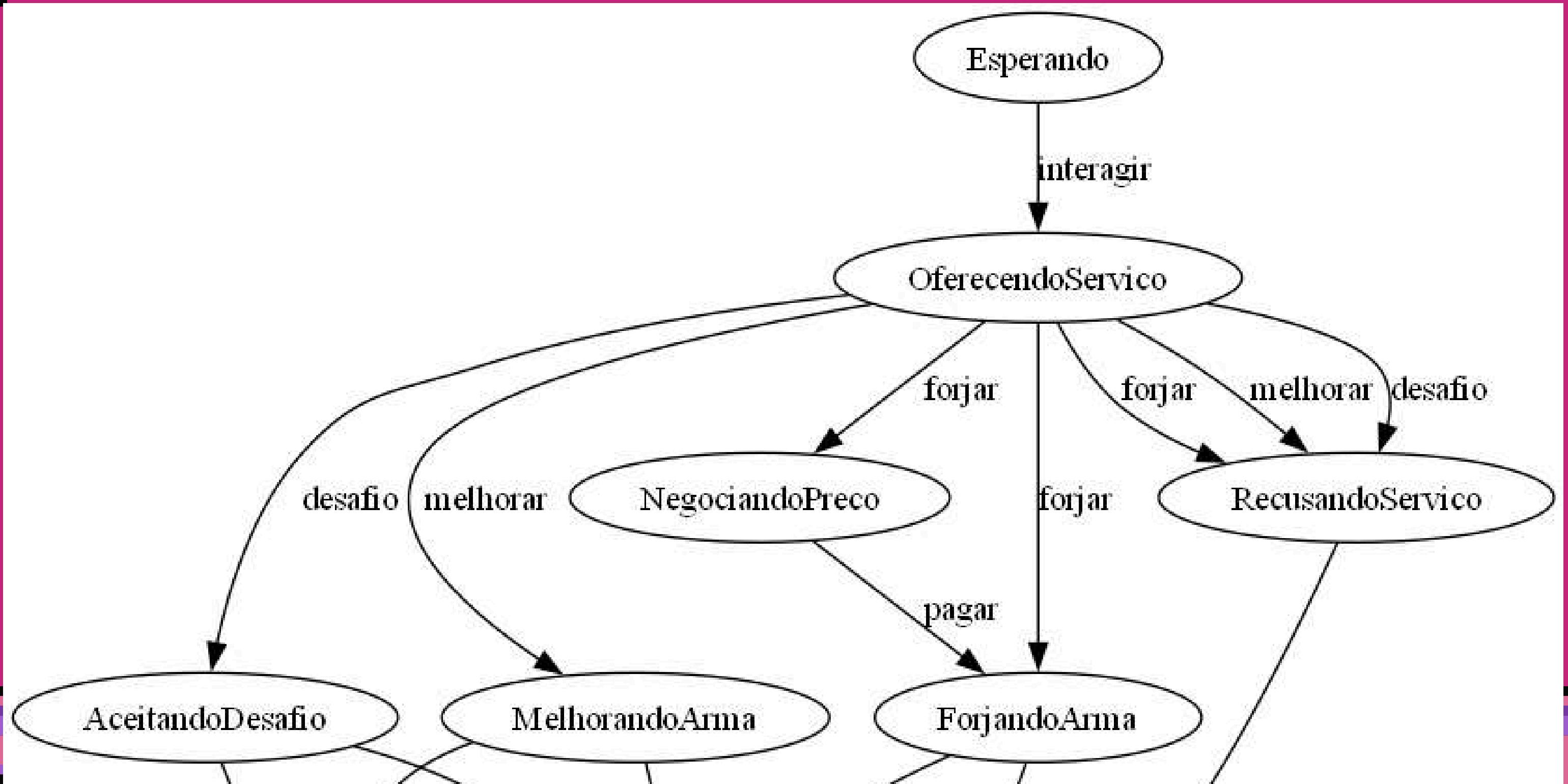
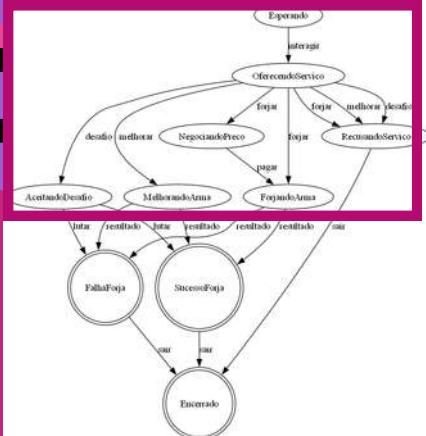
NPC MERCANTE- AFD



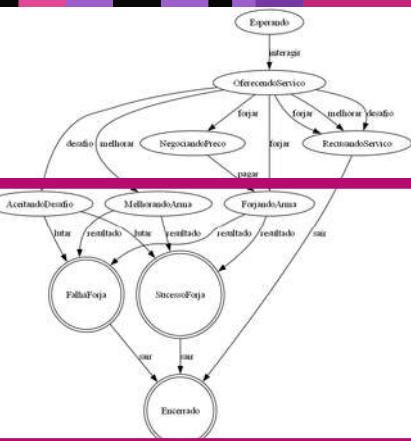
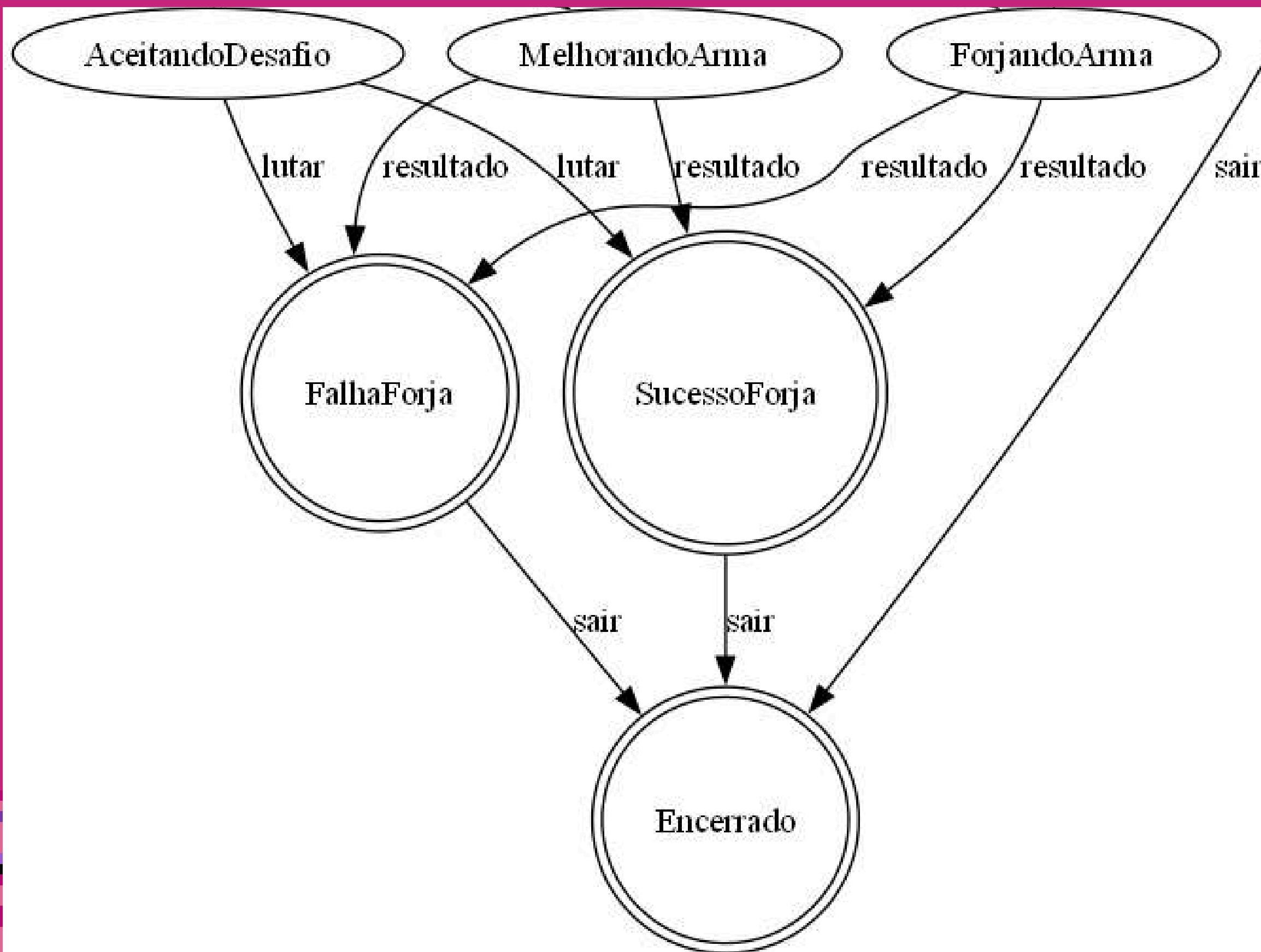
NPC FERREIRO - PSUEDO-AFN



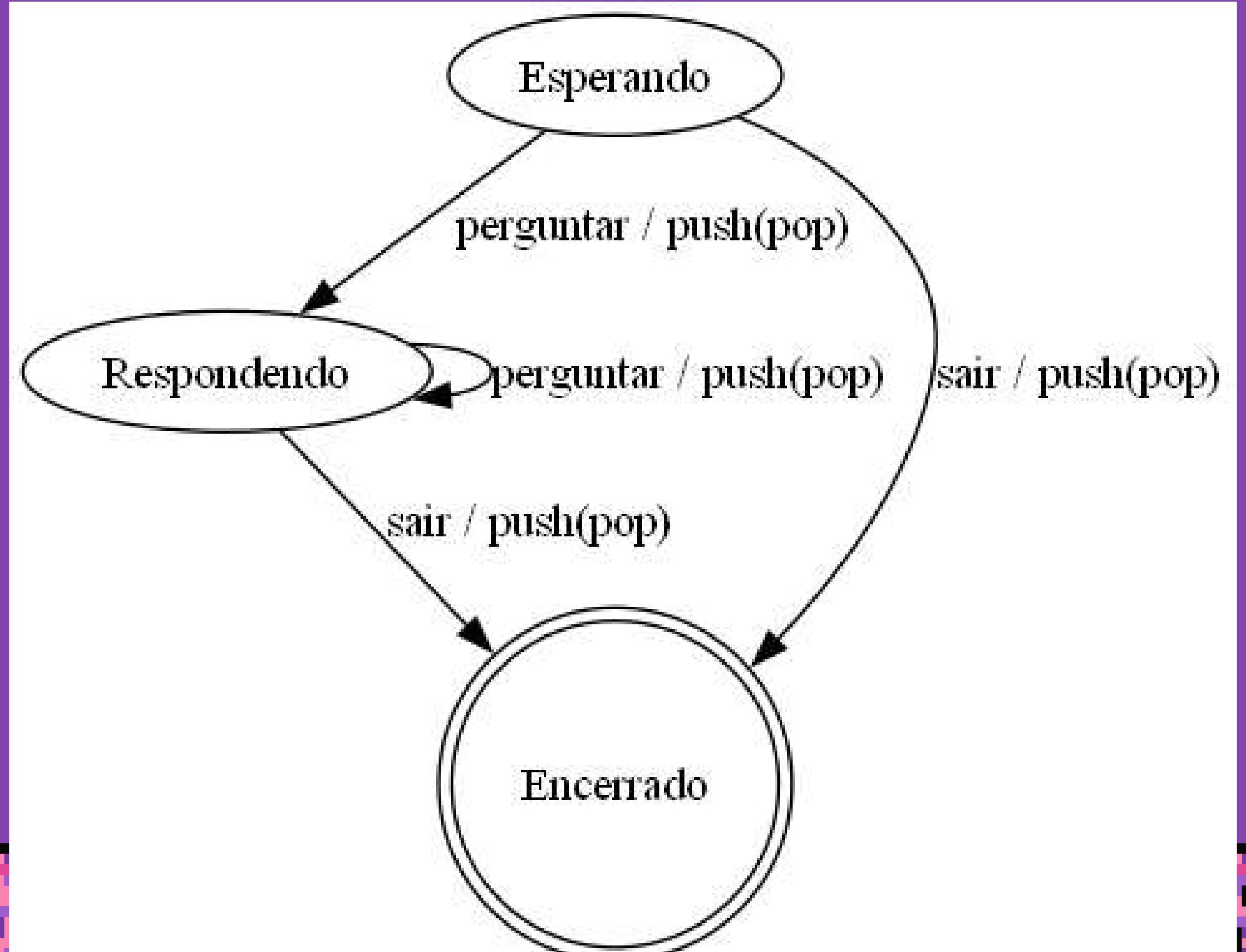
NPC FERREIRO-PSEUDO-AFN



NPC FERREIRO- PSUEDO-AFN



NPC INFORMANTE - AP



➡ VÍDEO DO PROGRAMA



ADENO: NPC COMPLEXO

Descrição

- ★ Simulação simples de um jogo RPG para implementação de NPC's baseados em automatos
- ★ Utilizamos a biblioteca Pyxel
- ★ Criamos um jogo 2D com um mapa básico, onde o jogador pode ser mover e interagir com elementos visuais
- ★ Foram implementados 3 NPC's distintos com comportamentos únicos
- ★ Aqui foi implementada uma lógica de venda de itens para o NPC Vendedor



MENU



ARQUIVOS .PY

- 🐍 ferreiro_npc_automato.py
- 🐍 informante_npc_automato.py
- 🐍 vendedor_npc_automato.py
- 🐍 main.py
- 🐍 visualizador_grafo_ferreiro.py
- 🐍 visualizador_grafo_informante.py
- 🐍 visualizador_grafo_vendedor.py

MENU



ARQUIVOS .PY

```
 config.py
 game.py
 map_utils.py
 npc_base.py
 npc_ferreiro.py
 npc_informante.py
 npc_vendedor.py
 player.py
```

ARQUIVOS

- ★ `ferreiro_npc_automato.py`: descreve o comportamento e os serviços oferecidos por um NPC Ferreiro, focado em forja, melhoria de equipamentos e até desafios.
- ★ `informante_npc_automato.py`: modela a interação com um NPC Informante, que fornece informações sobre o mundo do jogo com base nas perguntas do jogador.
- ★ `vendedor_npc_automato.py`: define o fluxo completo de interação com um NPC Vendedor, permitindo ao jogador comprar, vender e negociar itens.

ARQUIVOS

- ★ config.py: atua como o centralizador de todas as configurações e dados estáticos do jogo. Ele define parâmetros globais que são usados em múltiplas partes da aplicação.
- ★ game.py: classe principal do jogo, responsável por inicializar o ambiente Pyxel, gerenciar o loop principal do jogo (update e draw), controlar o jogador, os NPCs e as interações.
- ★ map_utils.py: contém funções utilitárias relacionadas à lógica do mapa e posições, como detecção de bloqueios e proximidade entre entidades.
- ★ main.py: é o ponto de entrada principal do jogo. Sua única função é iniciar a aplicação.

ARQUIVOS

- ★ `npc_base.py`: define as propriedades e comportamentos comuns que qualquer NPC que interage com o jogador deve ter, especialmente no que diz respeito ao sistema de diálogo baseado em autômatos.
- ★ `npc_informante.py`: focado em fornecer informações e "lore" do mundo do jogo ao jogador por meio de perguntas e respostas.
- ★ `npc_ferreiro.py`: estende a NPCBase para oferecer serviços de forja, melhoria de equipamentos e desafios, com lógicas de sucesso/falha e custo de recursos.
- ★ `npc_vendedor.py`: gerencia todas as interações de compra e venda de itens com o jogador, incluindo negociação e reações a ameaças.
- ★ `player.py`: representa o personagem jogável, controlando seu movimento, inventário e ouro.

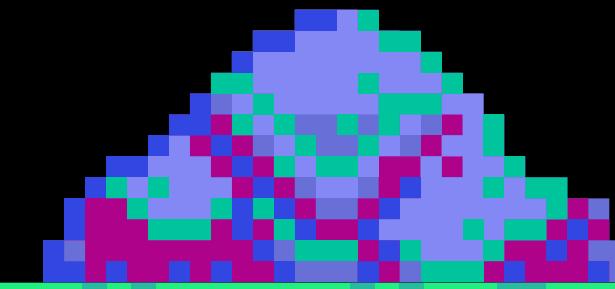
ARQUIVOS

- visualizador_grafo_ferreiro.py: cria um diagrama visual do autômato do Ferreiro
- visualizador_grafo_informante.py: gera um diagrama visual do autômato do Informante
- visualizador_grafo_vendedor.py: cria um diagrama visual do autômato do Vendedor

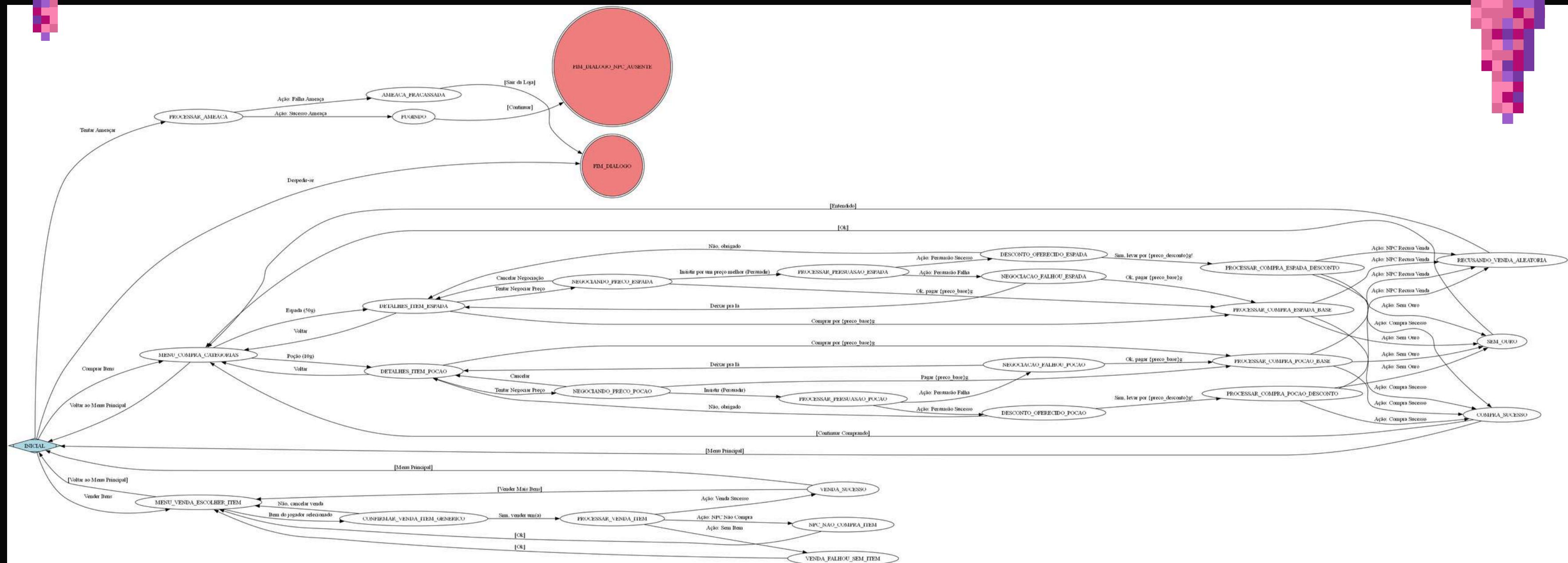
DIAGRAMAS



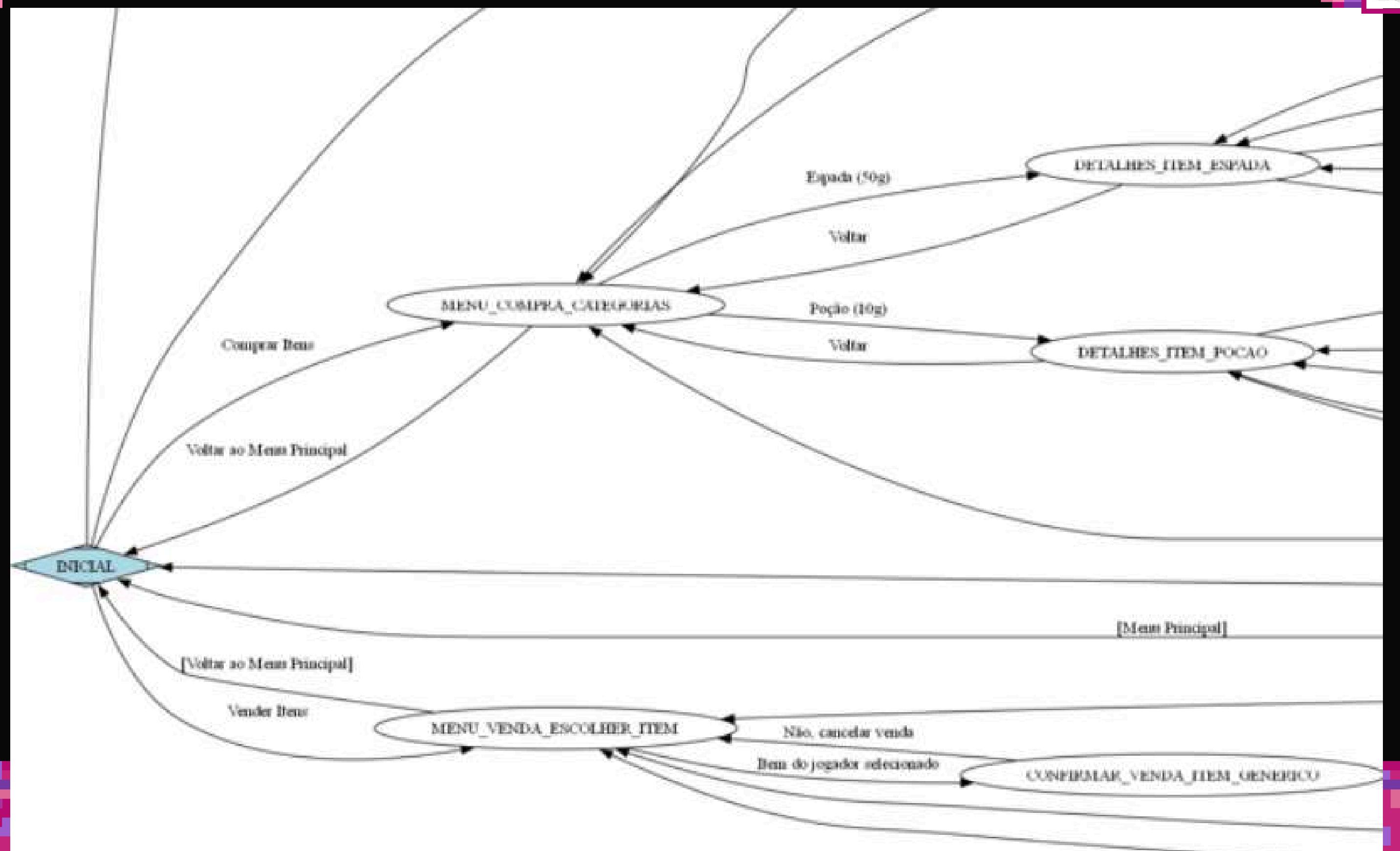
Representação Gráfica dos Estados e Transições



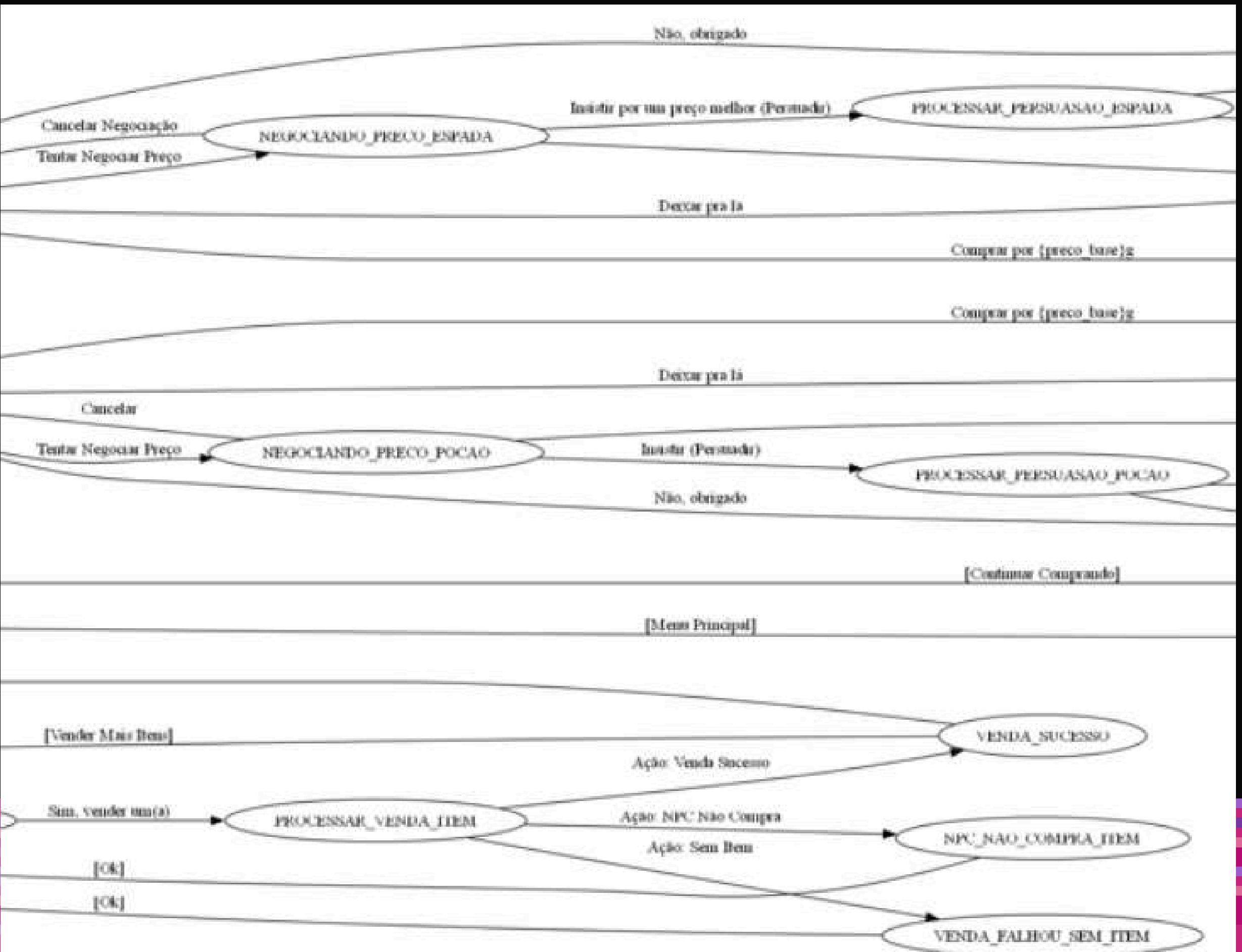
NPC VENDEDOR - AFD



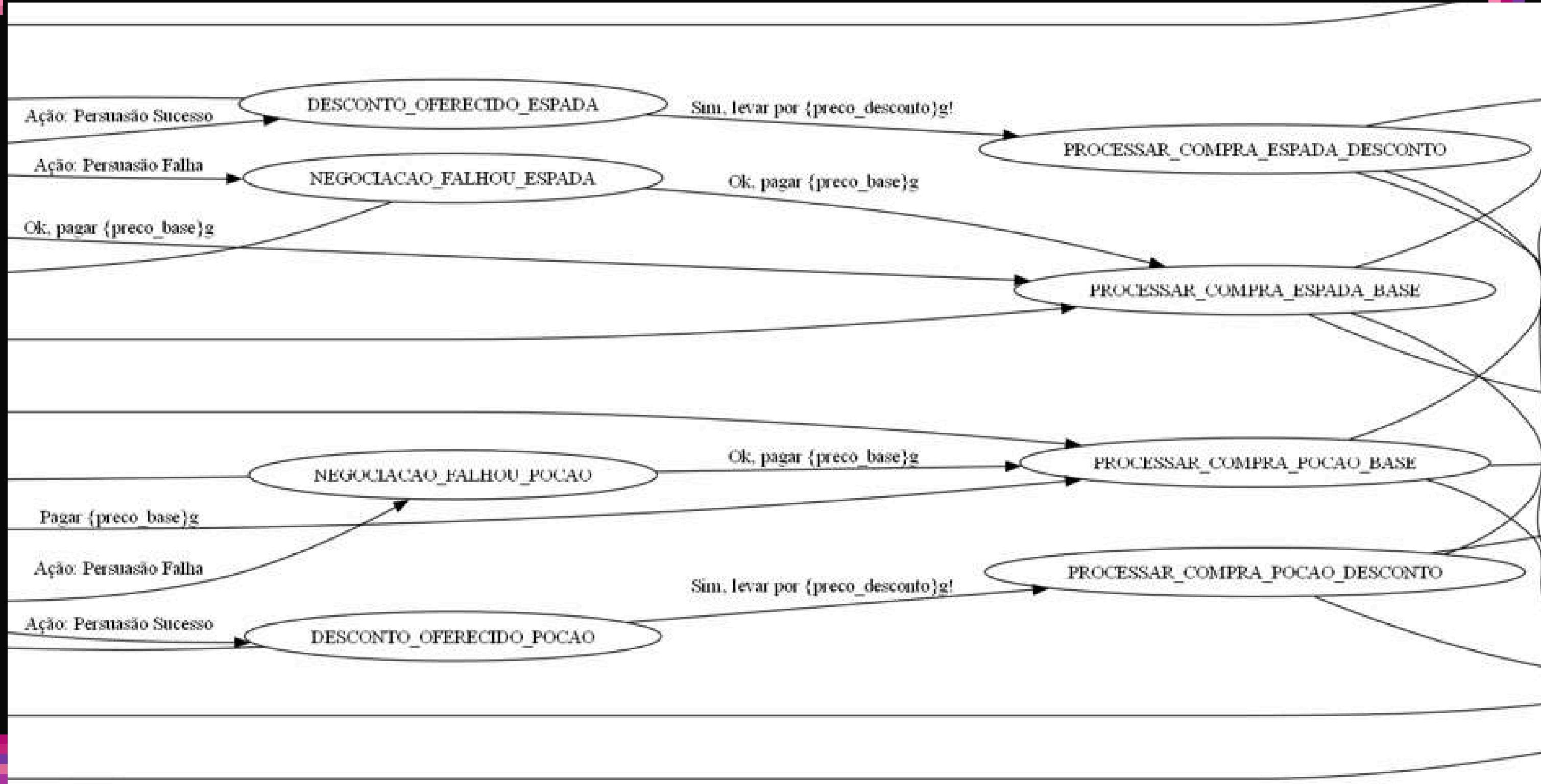
NPC VENDEDOR - AFD



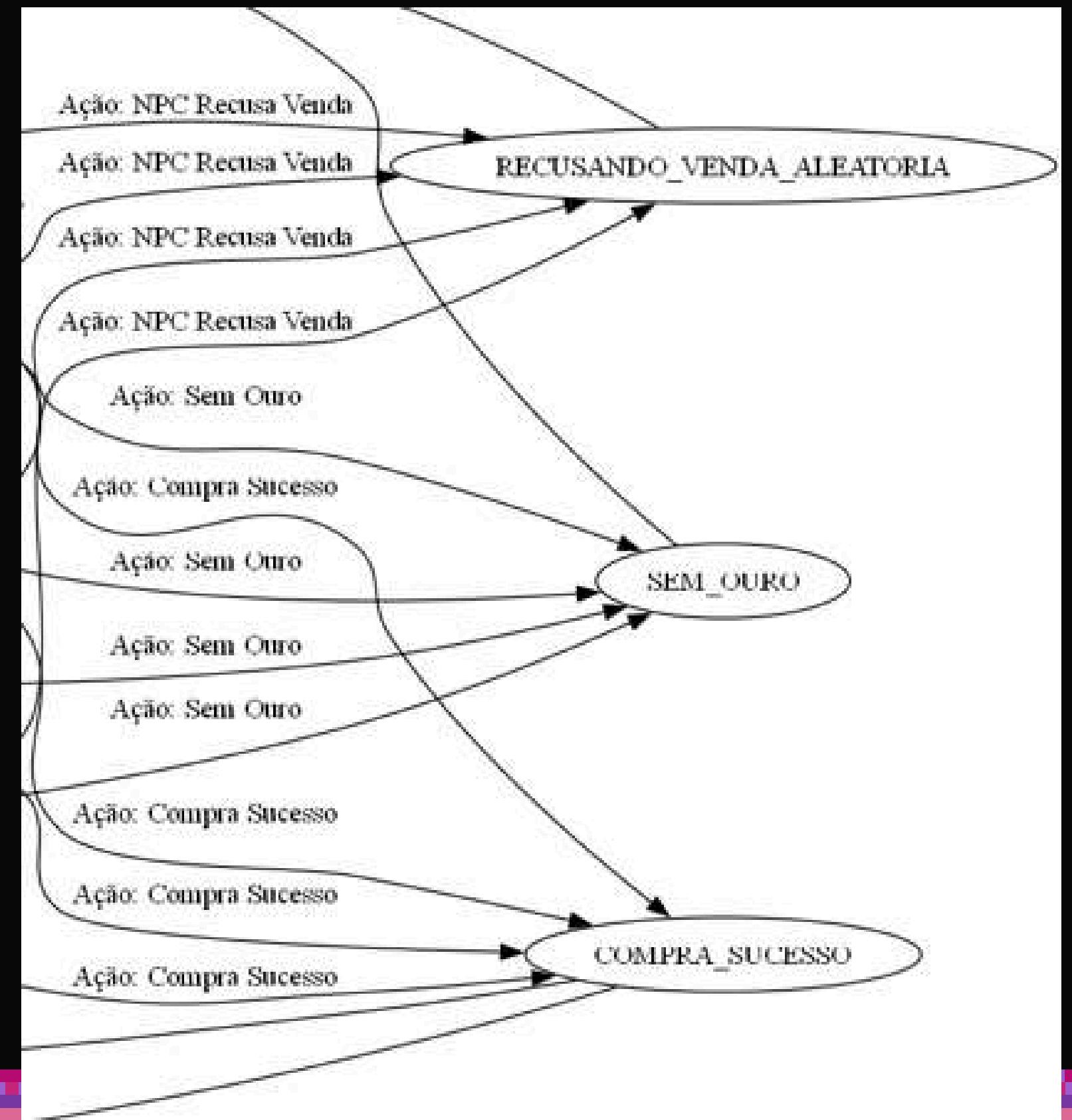
NPC VENDEDOR - AFO



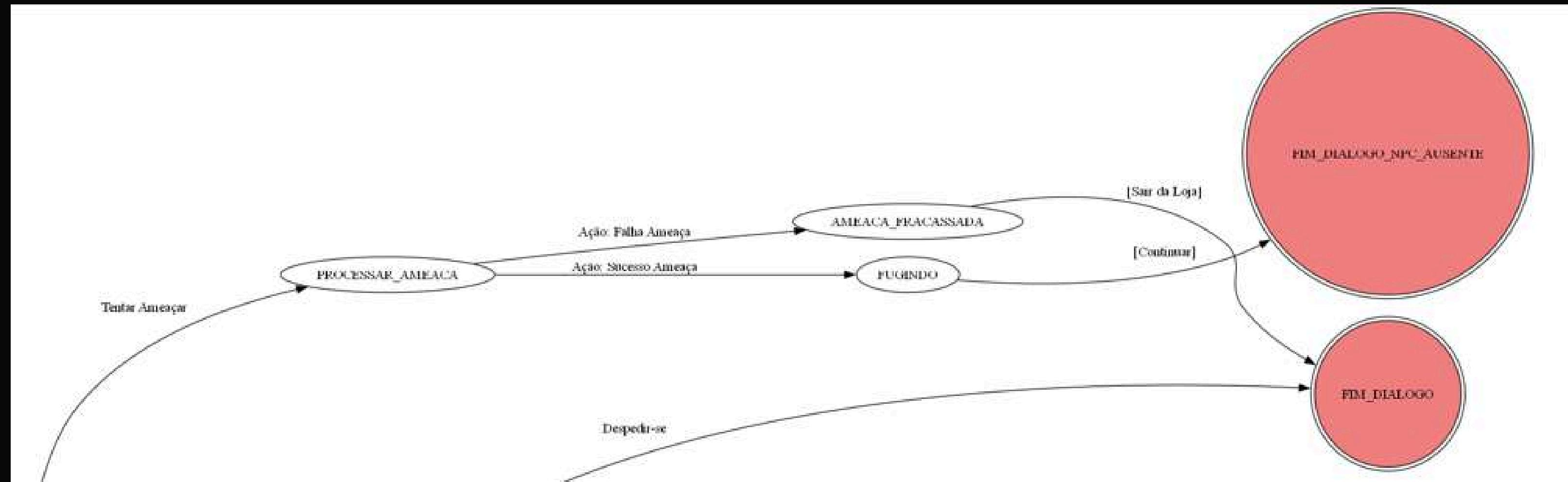
NPC VENDEDOR - AFO



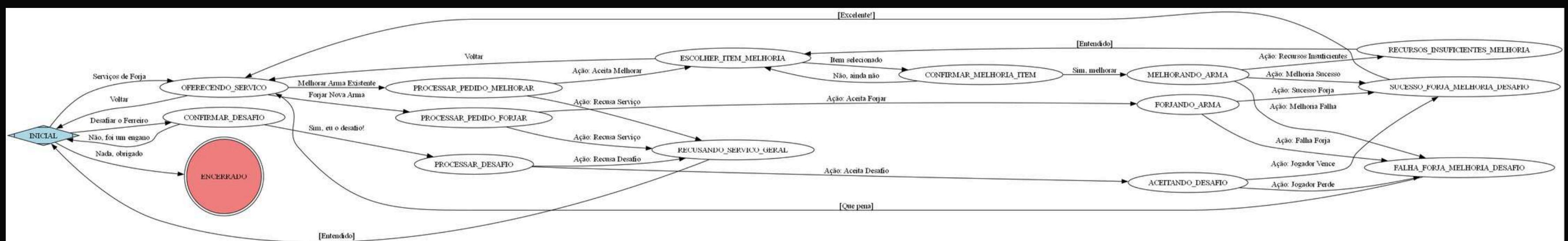
NPC VENDEDOR - AFD



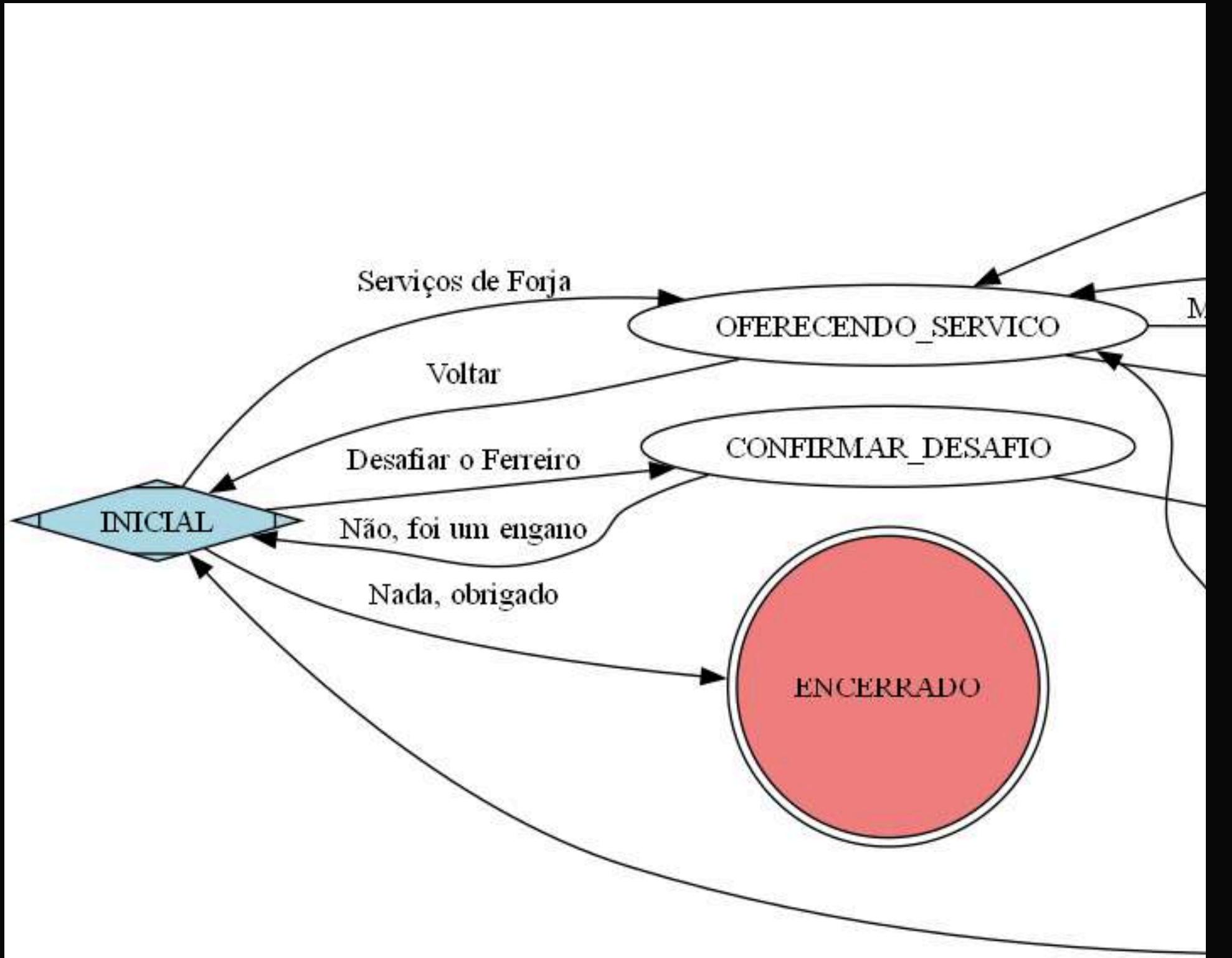
NPC VENDEDOR - AFD



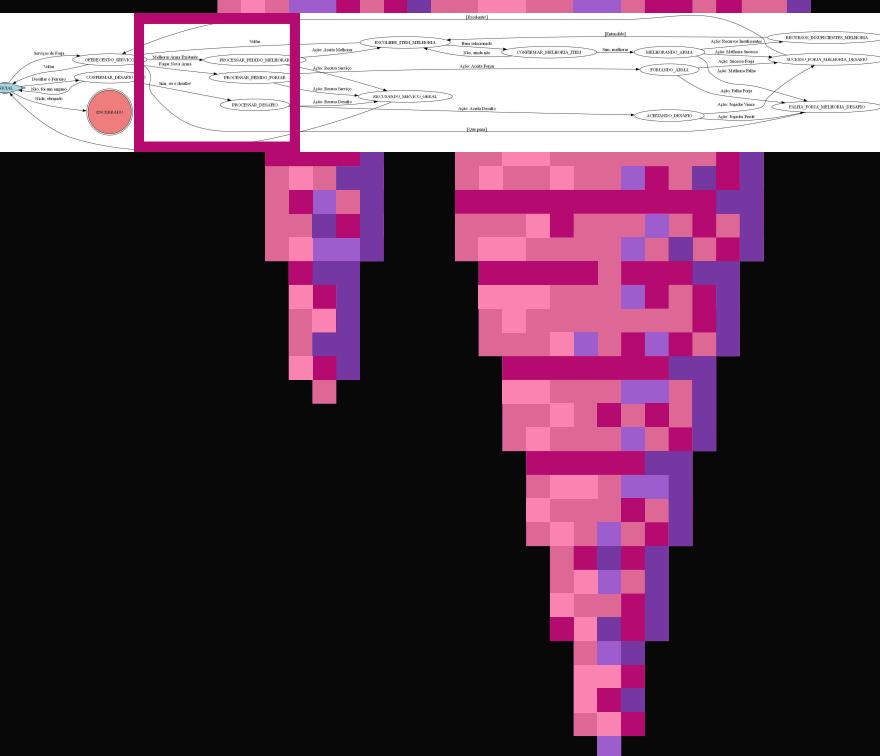
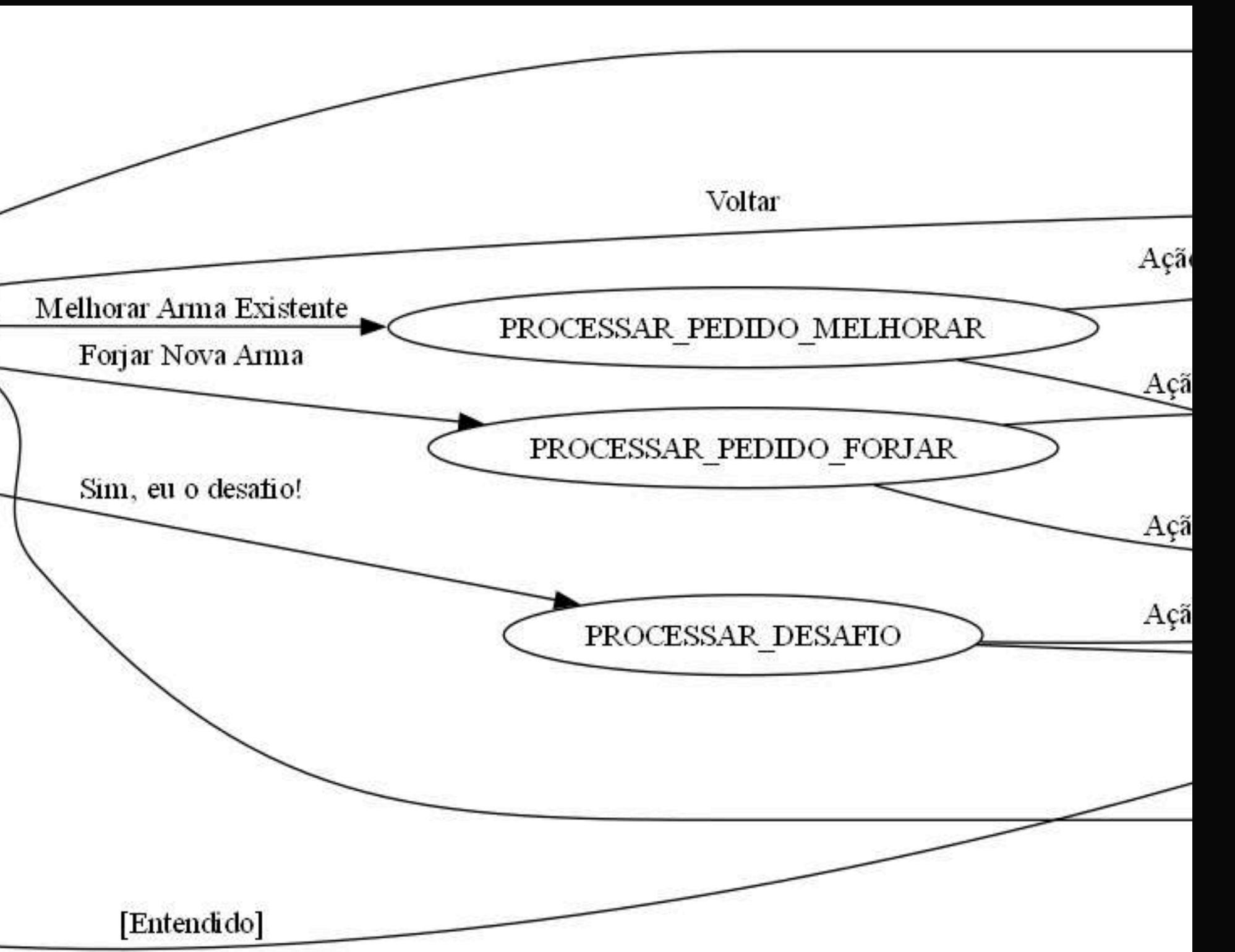
NPC FERREIRO- AFO



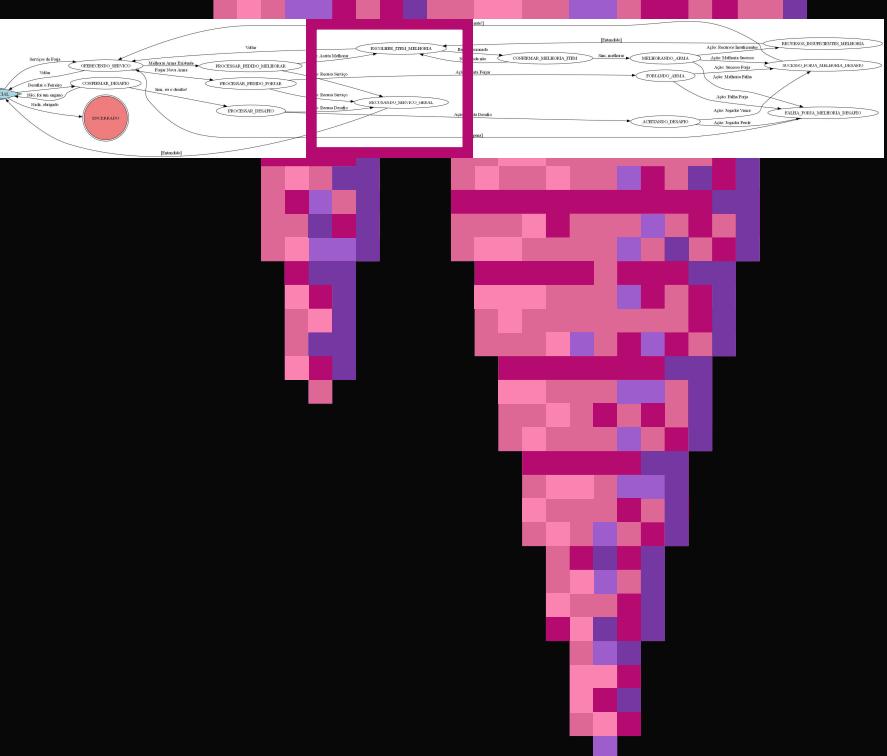
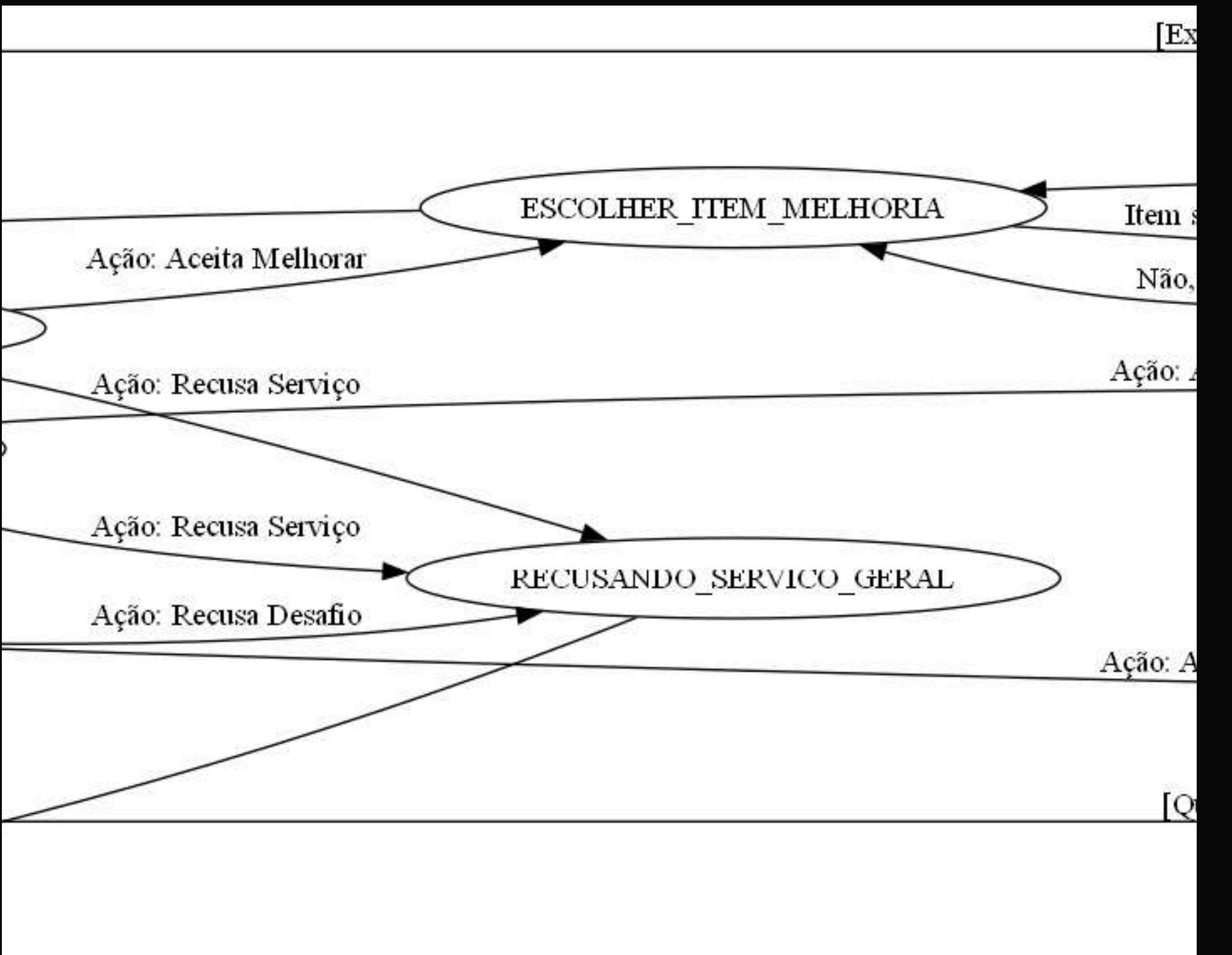
NPC FERREIRO- AFO



NPC FERREIRO - AFD



NPC FERREIRO- AFD



NPC FERREIRO - AFD

[Excelente!]

Entendido

Item selecionado

Não, ainda não

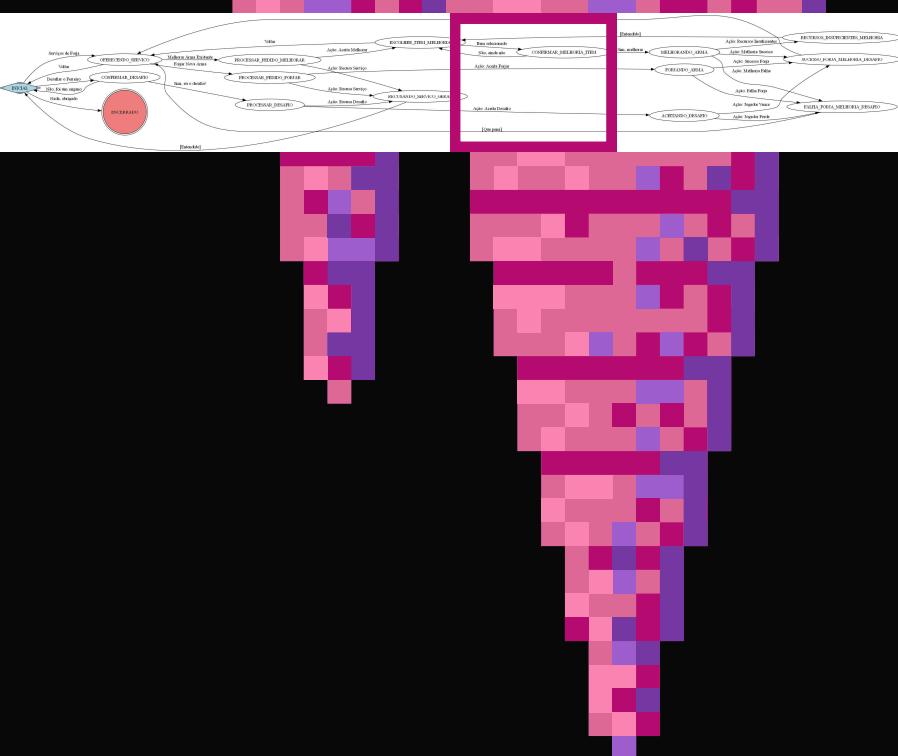
CONFIRMAR MELHORIA ITEM

Sim, melhor

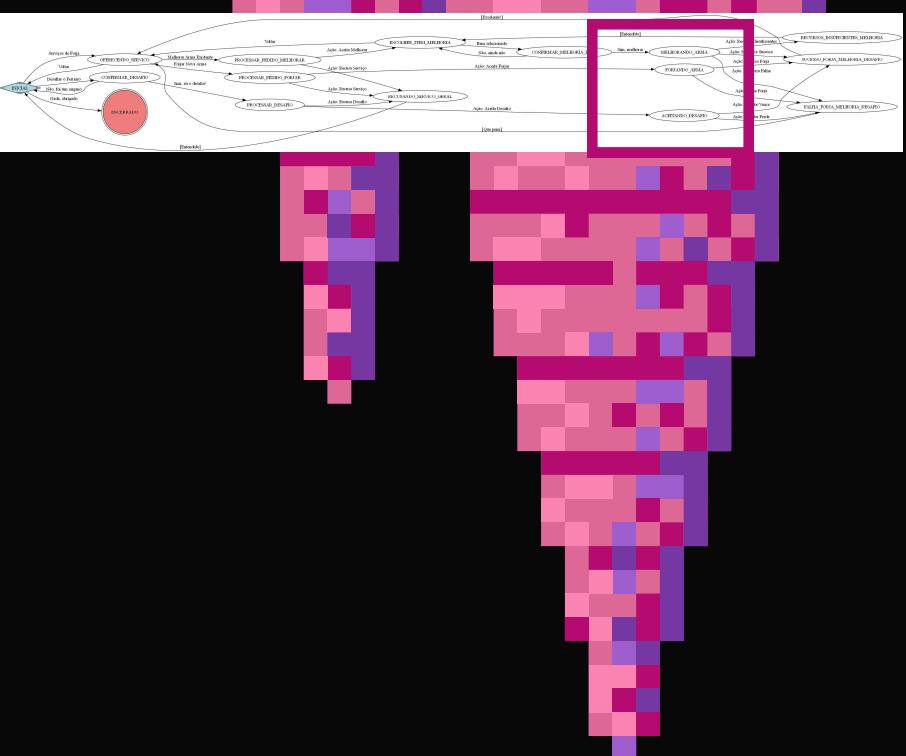
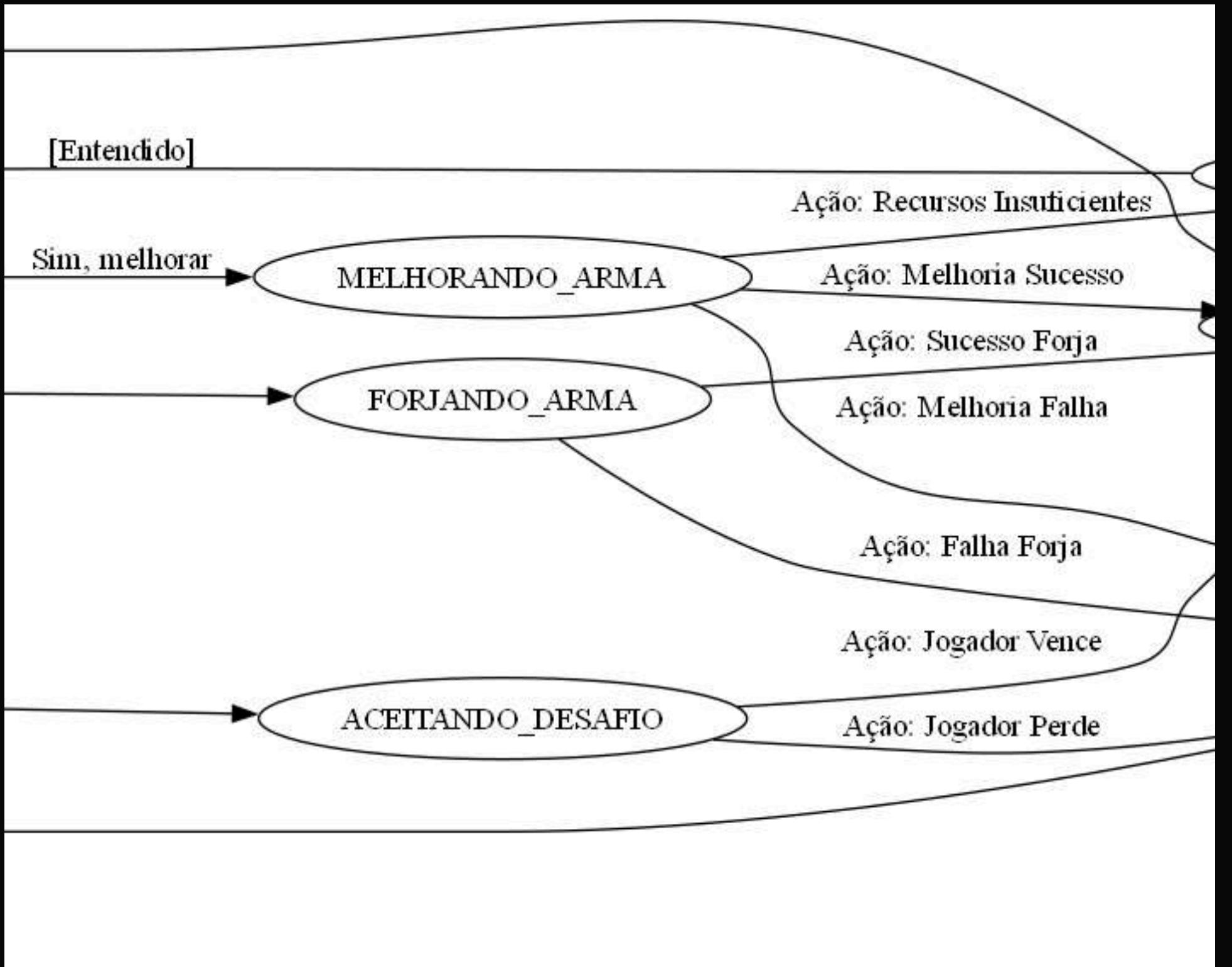
Ação: Aceita Forjar

Ação: Aceita Desafio

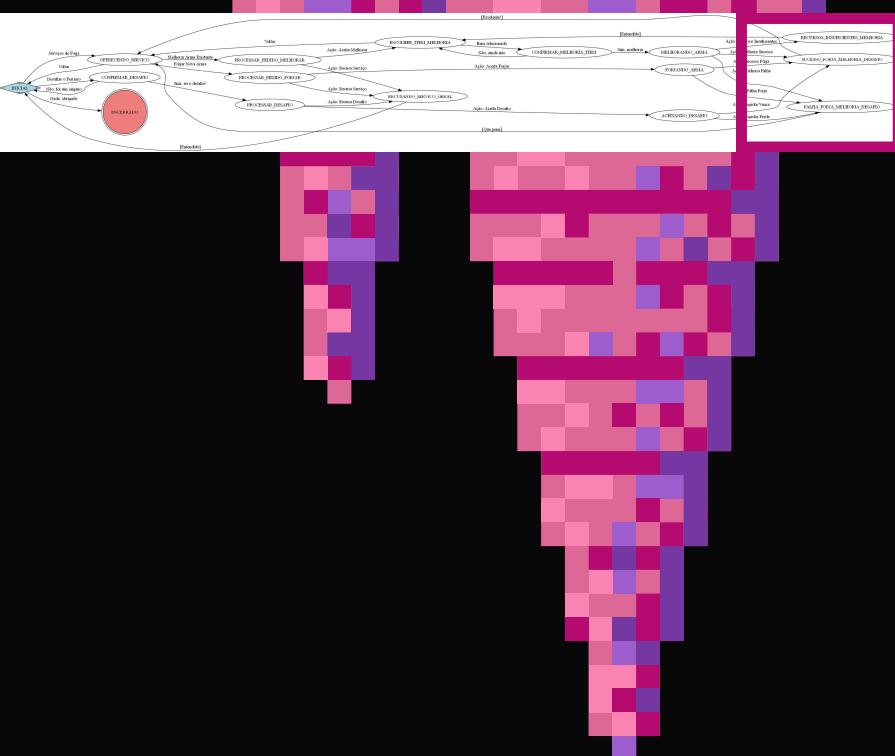
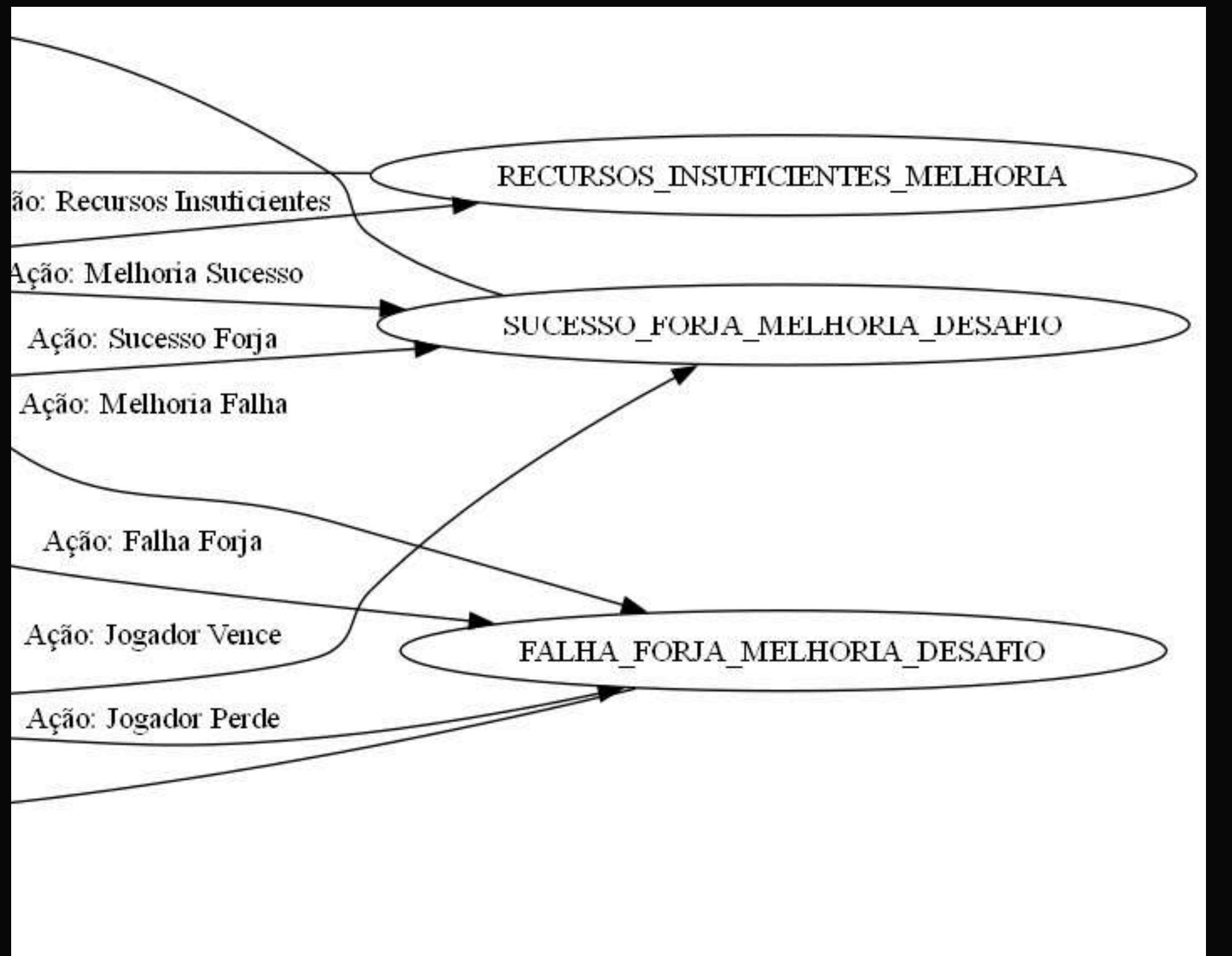
[Que pena]



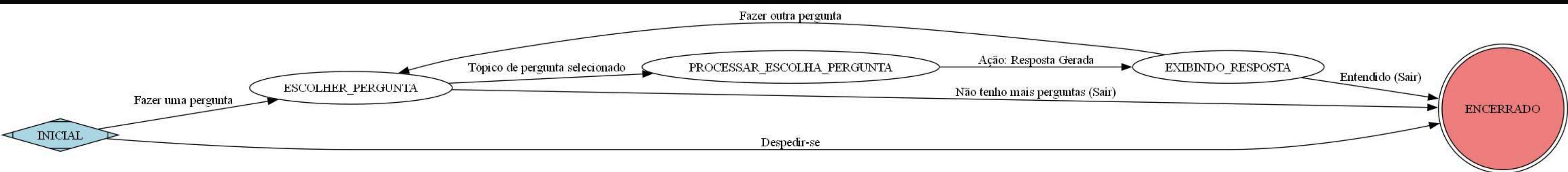
NPC FERREIRO- AFO



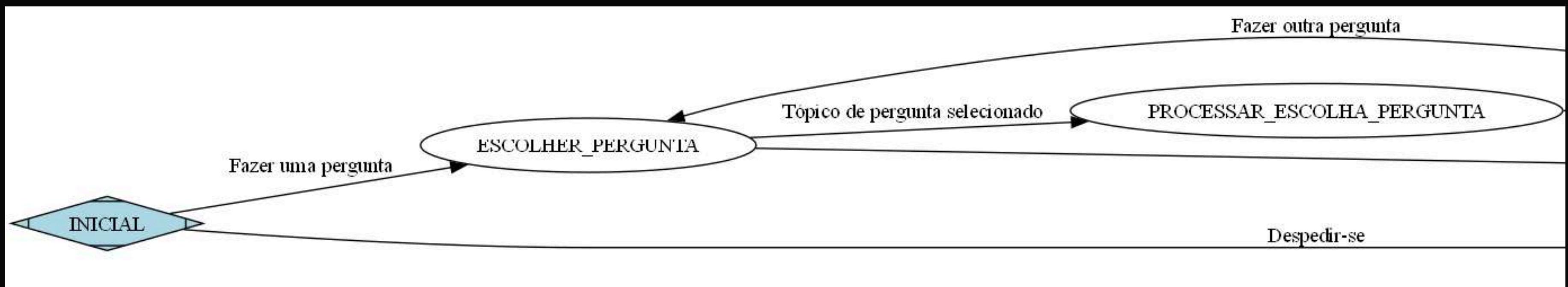
NPC FERREIRO - AFO



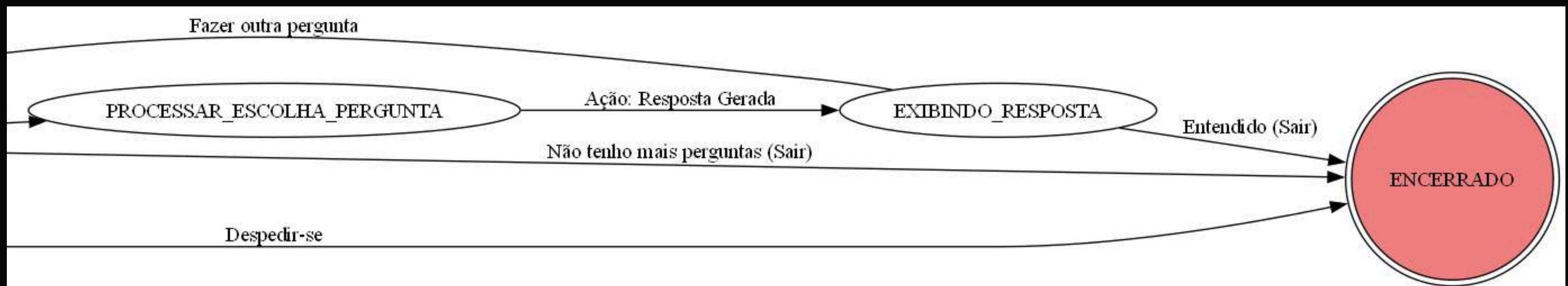
NPC INFORMANTE- AFD



NPC INFORMANTE- AFD



NPC INFORMANTE- AFD



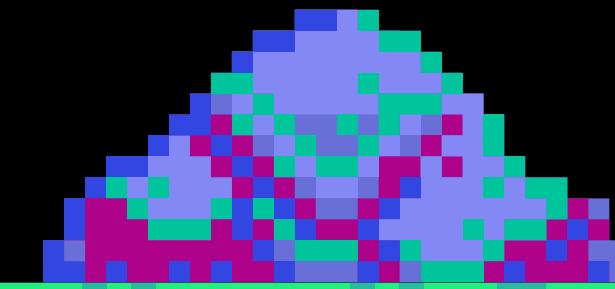
◀ VÍDEO DO PROGRAMA

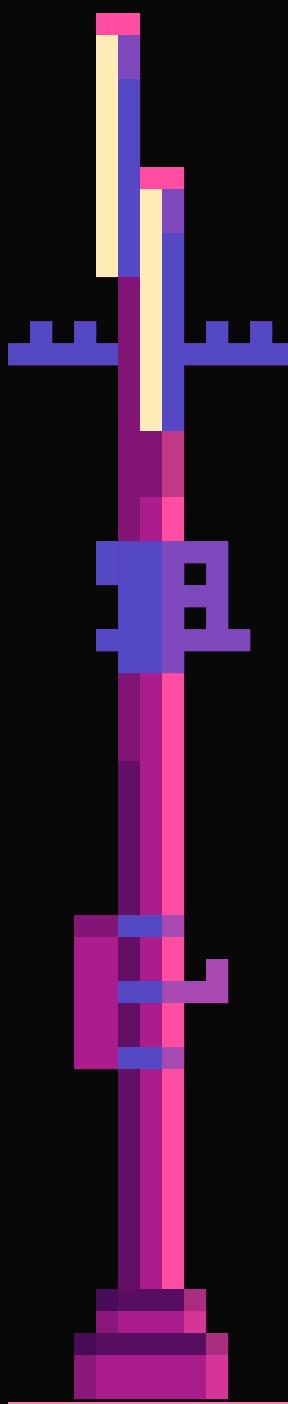


OBSERVAÇÕES



“Handlers de Ação”





OBSERVAÇÕES

- ★ Sabemos que um AFD é, por definição, totalmente previsível. Dada uma entrada e um estado atual, há sempre e apenas um próximo estado possível.
- ★ Os autômatos de nossos NPCs, são estruturalmente AFD's. As transições de estado são sempre claras e únicas para cada escolha do jogador.
- ★ A aparente "aleatoriedade" ou "probabilidade" de sucesso que observamos em algumas interações (como uma ameaça, uma persuasão ou uma forja) não reside nas transições do autômato em si, mas sim na lógica interna dos 'handlers de ação'.



OBSERVAÇÕES

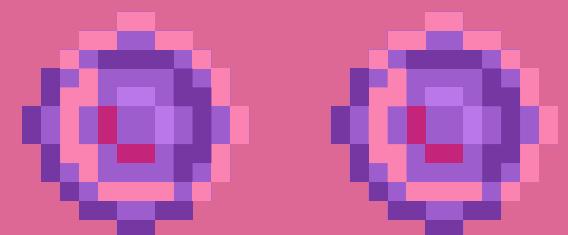
- ★ Quando o autômato atinge um estado que chama um action_handler (como PROCESSAR_AMEACA no Vendedor ou FORJANDO_ARMA no Ferreiro), é dentro desse handler que um cálculo de probabilidade é realizado.
- ★ O resultado da probabilidade (sucesso ou falha, por exemplo) determina qual será o próximo estado específico para o qual o autômato irá transicionar em seguida.
- ★ Dessa forma, mesmo que o resultado do action_handler seja probabilístico, o autômato continua sendo determinístico.

OBSERVAÇÕES

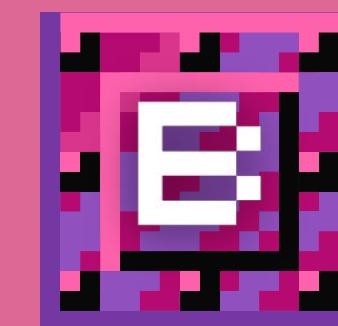
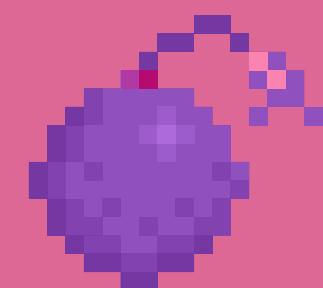
- ★ O `action_handler` em si é uma função determinística: dadas as mesmas entradas (incluindo o resultado da rolagem de dados interna), ela sempre produzirá o mesmo resultado.
- ★ A transição do autômato para o próximo estado resultante é sempre única. Se a rolagem de dados para a ameaça resultar em "sucesso", o autômato sempre vai para o estado 'FUGINDO'. Não há outra opção para esse resultado específico.
- ★ Assim, temos autômatos totalmente determinísticos para nossos NPCs, encapsulando a complexidade e a probabilidade dos eventos de jogo dentro dos 'handlers de ação'.

LIMITAÇÕES

ARKANOID

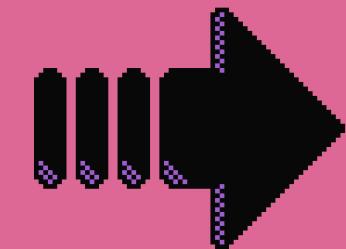


SIMPLICIDADE
DOS ESTADOS

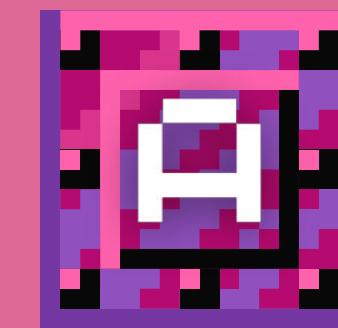
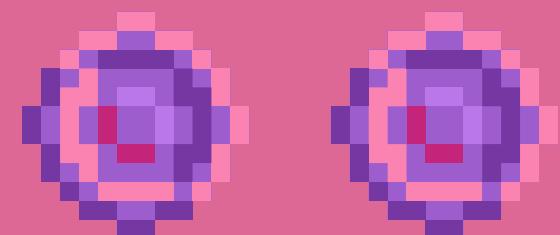


AUSÊNCIA DE
INTELIGÊNCIA
ADAPTATIVA

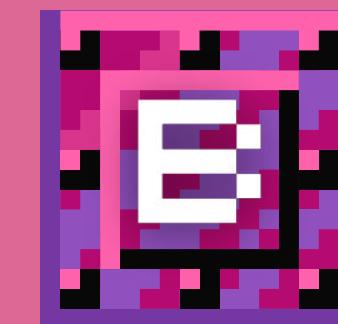
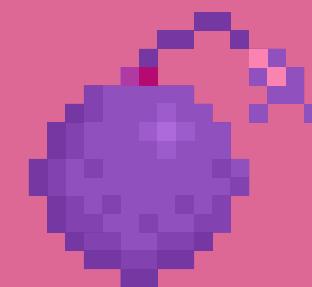
PRÓXIMAS
LIMITAÇÕES



ARKANOID



POWER-UPS
LIMITADOS



VELOCIDADE E
DINÂMICA FIXA

NPC

Limitações do 2º Jogo

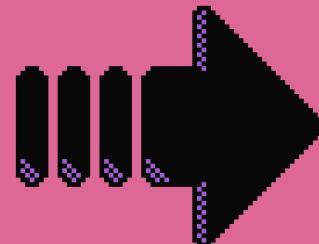


FALTA DE MEMÓRIA LONGA
OU HISTÓRICO DO JOGADOR



FALTA DE
IMPROVISACÃO
E ADAPTAÇÃO

PRÓXIMAS
LIMITAÇÕES



NPC

H PERSONALIDADE
LIMITADA

B AUSÊNCIA DE MEMÓRIA
CONTEXTUAL REAL

CONCLUSÃO

Opiniões Finais



CORRETO



INCORRETO

CONCLUSÃO

• O USO DE UM AUTÔMATO FINITO DETERMINÍSTICO (AFD) PERMITIU MODELAR O COMPORTAMENTO ESTRATÉGICO DO JOGADOR DE FORMA CLARA E CONTROLADA

• O AUTÔMATO FACILITOU A DEPURAÇÃO E ORGANIZAÇÃO DAS LÓGICAS DE COMPORTAMENTO, PROMOVENDO CLAREZA NO DESENVOLVIMENTO DO PADDLE CONTROLADO

• O PROJETO DEMONSTROU A APLICABILIDADE DOS AUTÔMATOS NO CONTROLE DE AGENTES EM JOGOS, ALÉM DE REFORÇAR A CONEXÃO ENTRE TEORIA DA COMPUTAÇÃO E APLICAÇÕES PRÁTICAS



01



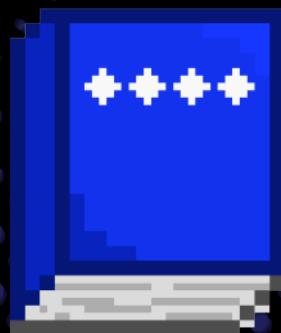
07



12



CONCLUSÃO



DEMONSTRAMOS COMO AUTÔMATOS FINITOS DETERMINÍSTICOS (AFD) SÃO UMA SOLUÇÃO EFICIENTE E ORGANIZADA PARA MODELAR E GERENCIAR FLUXOS DE DIÁLOGO E COMPORTAMENTOS DE NPCS EM JOGOS.

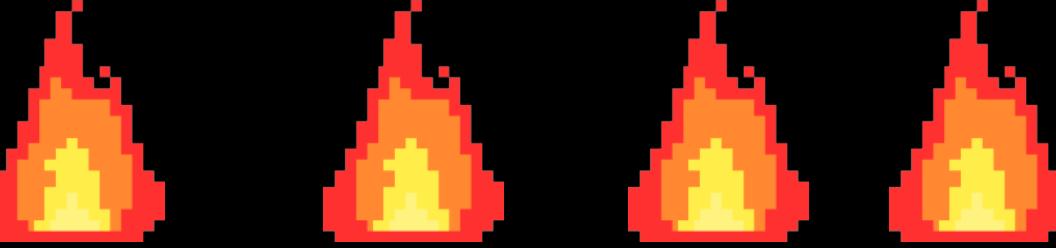
ELABORAMOS A INTEGRAÇÃO DE LÓGICA DE JOGO (CÁLCULO DE OURO, INVENTÁRIO, CHANCES DE SUCESSO) DIRETAMENTE NO FLUXO DO AUTÔMATO USANDO "HANDLERS DE AÇÃO" E "GERADORES DE OPÇÕES".

A VISUALIZAÇÃO DOS GRAFOS SE MOSTROU CRUCIAL PARA A COMPREENSÃO E DEPURAÇÃO DOS FLUXOS DE CONVERSA.

MENU



MUITO
OBRIGADO

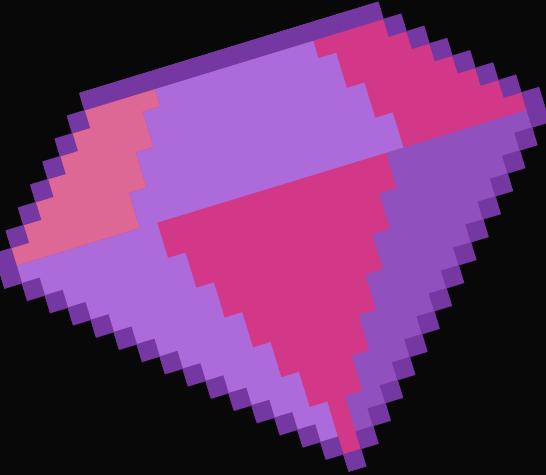


REFERÊNCIAS

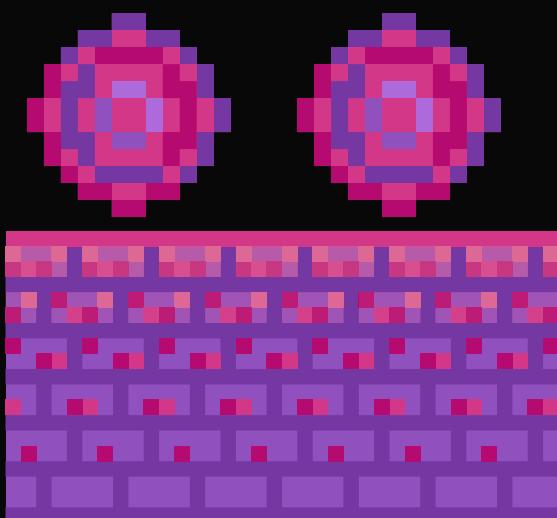
 John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman – Introdução à Teoria de Autômatos, Linguagens e Computação (2^a Edição).

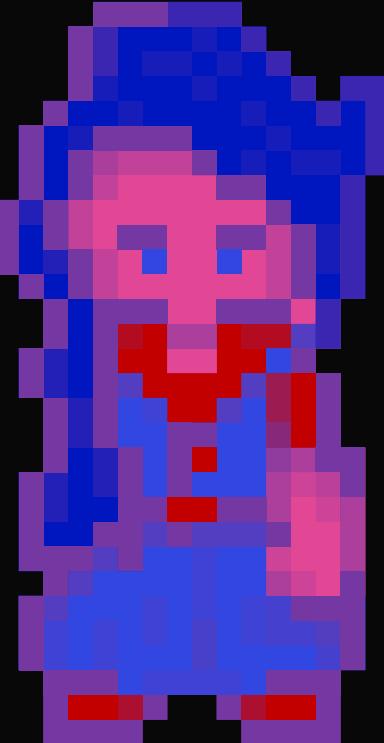
 PYXEL. Pyxel: Retro Game Engine for Python. Disponível em: <https://github.com/kitao/pyxel>. Acesso em: 3 jun. 2025.

 WIKIPÉDIA. Teoria dos autômatos. Wikipédia, a enciclopédia livre, 2024. Disponível em: https://pt.wikipedia.org/wiki/Teoria_dos_aut%C3%A3omatos. Acesso em: 3 jun. 2025.



REFERÉNCIAS



- 
-  HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation*. 3. ed. Boston: Pearson, 2006.
 -  Aguiar, M. and Mahajan, S. 2010. "Monoidal Functors, Species, and Hopf Algebras".
 -  John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman (2000). *Introduction to Automata Theory, Languages, and Computation* (2nd Edition). [S.l.]: Pearson Education. ISBN 0-201-44124-1