ESPECIFICAÇÃO FORMAL

INTRODUÇÃO

Este documento apresenta a especificação formal dos requisitos do jogo Eco Runner.

PLANEJAMENTO

A especificação formal dos requisitos será realizada utilizando notação Z e redes de petri. Para cada requisito funcional e não funcional, haverá uma especificação.

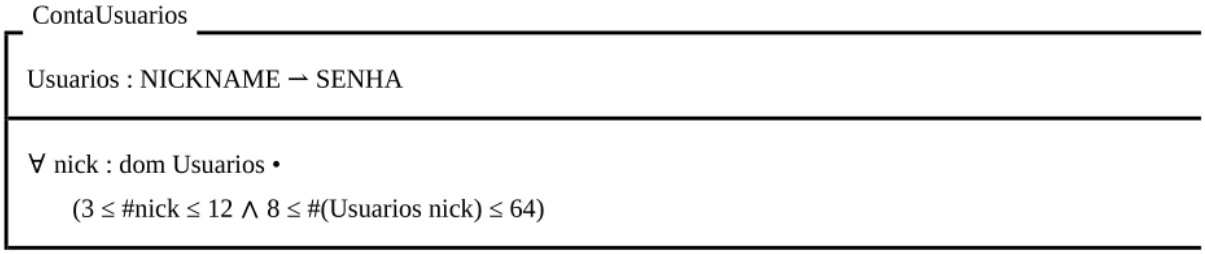
REQUISITOS FUNCIONAIS

RFO1

- Notação Z
- Tipos Básicos:

  [NICKNAME, SENHA]

  RespostaLogin :: = valido | erro_nick_invalido | erro_senha_invalida



- Esquema de estado:

- Esquemas de operação:

LoginSucesso

ΞContaUsuarios
nick? : NICKNAME
senha? : SENHA
resp! : RespostaLogin

---

nick? ∈ dom Usuarios
Usuarios(nick?) = senha?
resp! = valido

NickInvalido

ΞContaUsuarios
nick? : NICKNAME
senha? : SENHA
resp! : RespostaLogin

---

nick? ∉ dom Usuarios
resp! = erro_nick_invalido

SenhaInvalida

ΞContaUsuarios
nick? : NICKNAME
senha? : SENHA
resp! : RespostaCadastro

---

nick? ∉ dom Usuarios
Usuarios nick ≠ senha?
resp!= erro_senha_invalida

- Operação completa:

  Login ≙ LoginSucesso ⩒ NickInvalido ⩒ SenhaInvalida

RFO2

- Notação Z
- Tipos Básicos:

  [NICKNAME, SENHA]

  RespostaCadastro :: = valido | erro_nick_existe | erro_formato

- Esquema de estado:

ContaUsuarios

$$Usuarios : NICKNAME \pfun SENHA$$

$$\forall nick : dom\ Usuarios \bullet (3 \leq \#nick \leq 12 \land 8 \leq \#(Usuarios\ nick) \leq 64)$$

- Esquemas de operação:

CadastrarSucesso

ΔContaUsuarios
nick? : NICKNAME
senha? : SENHA
resp! : RespostaCadastro

$3 \leq \#nick? \leq 12$
$8 \leq \#senha? \leq 64$
$nick? \notin dom\ Usuarios$
$Usuarios' = Usuarios \cup \{nick? \mapsto senha?\}$
resp! = valido

FormatoInvalido

ΞContaUsuarios
nick? : NICKNAME
senha? : SENHA
resp! : RespostaCadastro

$(3 > \#nick? \lor \#nick? > 12) \lor$
$(8 > \#senha? \lor \#senha? > 64)$
resp! = erro_formato

NickJaExiste

ΞContaUsuarios
nick? : NICKNAME
senha? : SENHA
resp! : RespostaCadastro

$3 \leq \#nick? \leq 12$
$8 \leq \#senha? \leq 64$
$nick? \in dom\ Usuarios$ |
resp! = erro_nick_existe

- Operação completa:

  Cadastrar ≙ CadastrarSucesso ∨ FormatoInvalido ∨ NickJaExiste

RF04

- Notação Z
- Tipos básicos

  TIPO_LIXO : := papel | plastico | vidro | metal

  RespostaSort : : = valido | erro_vidas_zero | erro_inventario_vazio

- Esquema de estado:

EstadoFase2

vidas : ℕ
inventario : TIPO_LIXO ⇸ ℕ
lixeiras : TIPO_LIXO ⇸ ℕ
game_over : 𝔹
fase_completa : 𝔹

---

vidas ≤ 5
game_over ⇔ vidas = 0
∀ t : TIPO_LIXO •
  (lixeiras(t) ≤ 5 ∧ inventario(t) ≤ 5)
∀ t : TIPO_LIXO •
  (inventario(t) + lixeiras(t) = 5)
fase_completa ⇔ (∀ t : TIPO_LIXO • lixeiras(t) = 5)

- Esquemas de operação:

SortItemCorreto

ΔEstadoFase2
item? : TIPO_LIXO
lixeira? : TIPO_LIXO
resp! : RespostaSort

¬game_over ∧ ¬fase_completa
item? = lixeira?
inventario(item?) > 0
lixeiras(lixeira?) < 5

inventario' = inventario ⊕ {item? ↦ inventario(item?) - 1}
lixeiras' = lixeiras ⊕ {lixeira? ↦ lixeiras(lixeira?) + 1}
vidas' = vidas
Resp! = valido

SortItemErrado

ΔEstadoFase2
item? : TIPO_LIXO
lixeira? : TIPO_LIXO
resp! : RespostaSort

¬game_over ∧ ¬fase_completa
item? ≠ lixeira?
inventario(item?) > 0

vidas' = vidas - 1
inventario' = inventario
lixeiras' = lixeiras
resp! = valido

InventarioVazio

ΔEstadoFase2

item? : TIPO_LIXO

resp! : RespostaSort

---

inventario(item?) = 0

resp! = erro_inventario_vazio

JogoPerdido

ΞEstadoFase2

resp! : RespostaSort

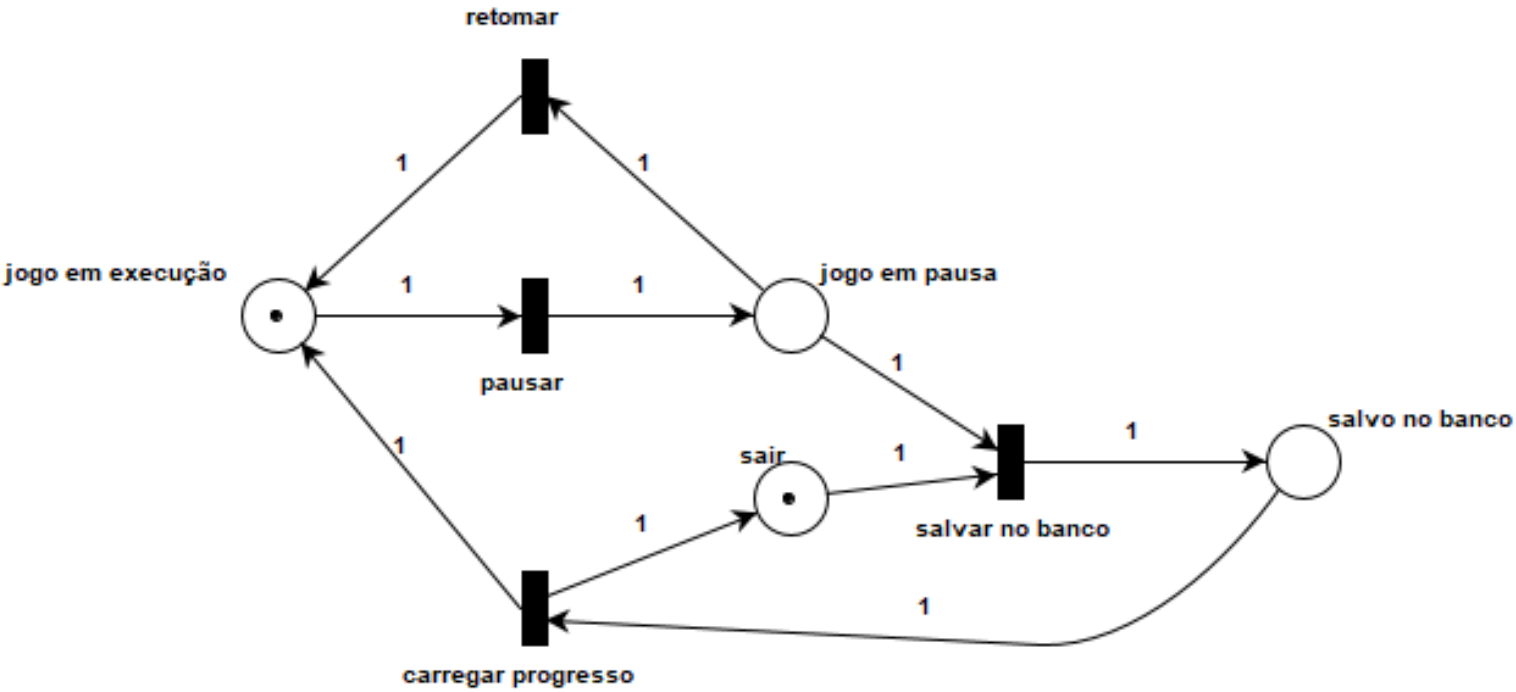
---

game_over

resp! = erro_vidas_zero

- Operação completa:
  SortItem ≙ SortItemCorreto ∨ SortItemErrado ∨ InventarioVazio ∨ JogoPerdido

RF06



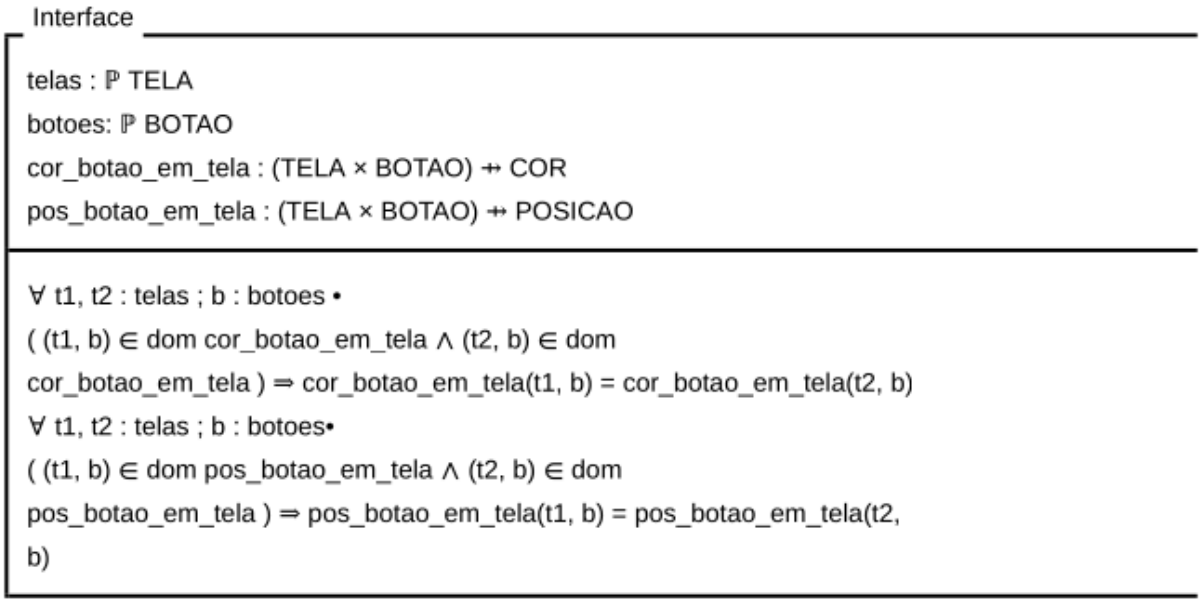
REQUISITOS NÃO FUNCIONAIS

RNF02

- Notação Z
- Tipos básicos

  [TELA, COR, POSICAO, BOTAO]

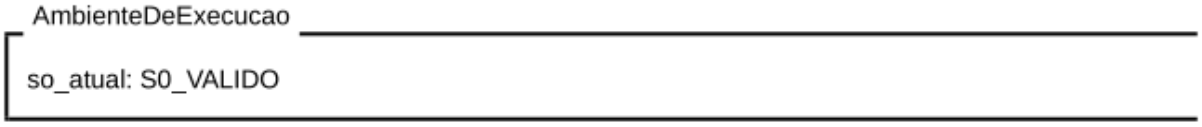
- Esquema de estado:



RNF03

- Notação Z
- Tipos básicos:
    - SO_VALIDO ::= windows10 | windows11 | ubuntu22_04

- Esquema de estado



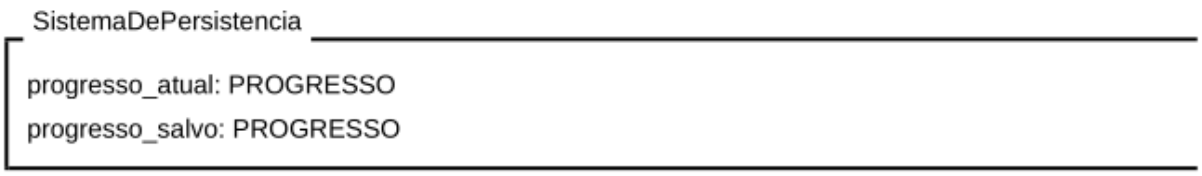
RNF04

- Notação Z
- Tipos básicos

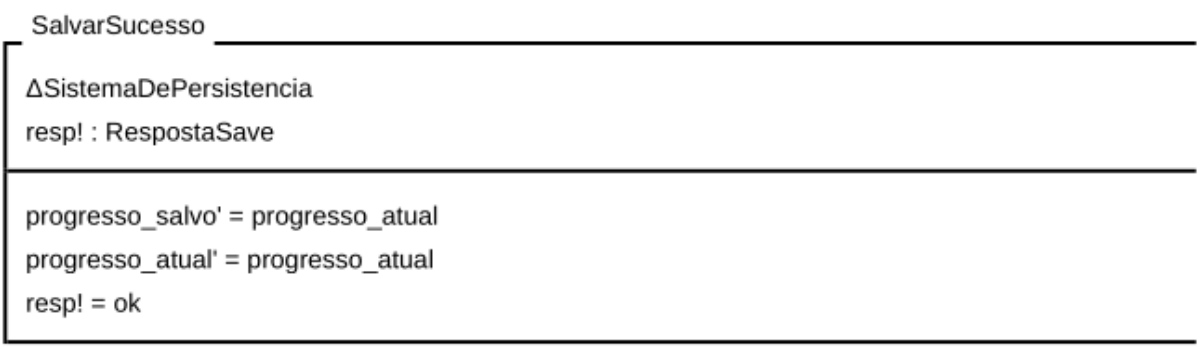
[PROGRESSO]

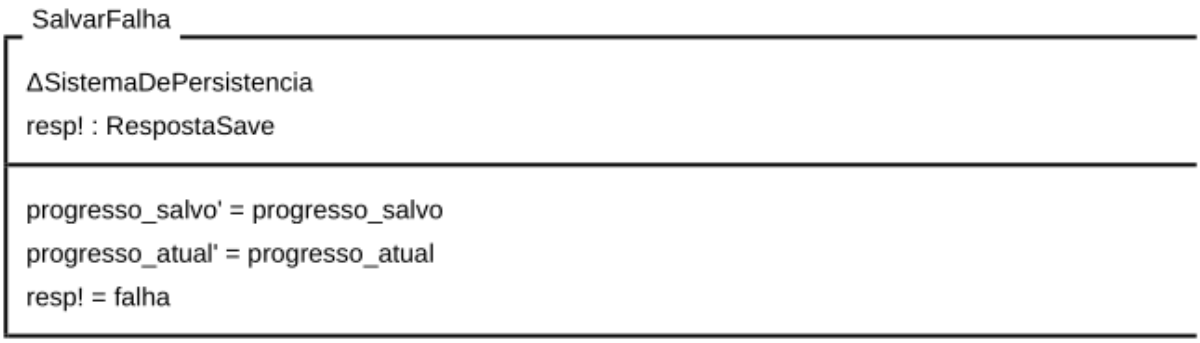
RespostaSave : := sucesso | falha

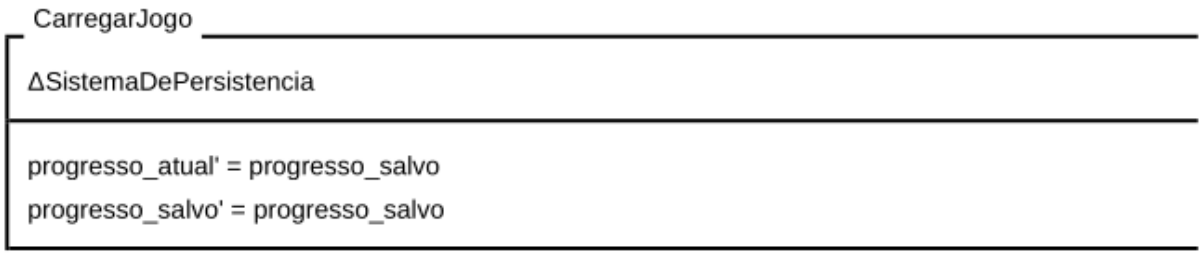
- Esquema de estado



- Esquemas de operação







- Operação completa

  $$SalvarJogo \triangleq SalvarSucesso \lor SalvarFalha$$