Projeto Go

# Especificação de Requisitos de Software

**Versão 1.0 - 03/04/2023**

| Versão | Autor(es) | Data | Ação |
| --- | --- | --- | --- |
| 1.0 | Leonardo Luiz, Leonardo Brito, Yuiti Kanekiyo | 03/04/2023 | Estabelecimento de Requisitos |

**Conteúdo:**

[**Especificação de Requisitos de Software 1**](#_1k8pb1u2kj4z)

[**1. Introdução 2**](#_45wqzr1cfq4l)

[1.1 Objetivo 2](#_b5psn2d6n1dc)

[1.2 Referências 2](#_mhwfb0sgb08z)

[**2.Visão geral 2**](#_u6wpbhqajqc7)

[2.1 Arquitetura do programa 2](#_ylam2oi40w4s)

[2.2 Premissas de desenvolvimento 2](#_qgvktu9qwhur)

[**3. Requisitos de Software 3**](#_5aw225f5epug)

[3.1 Requisitos funcionais 3](#_gg391mbh3ihu)

[3.2 Requisitos Não funcionais 5](#_ge31zsr5gv54)

[**Apêndice: Regras do jogo Go 5**](#_e2rhust67y1g)

# 1. Introdução

## 1.1 Objetivo

Desenvolvimento de um programa para promover a disputa do jogo Go entre dois jogadores.

## 1.2 Referências

https://baduk.com.br/regras-basicas/

https://pt.wikipedia.org/wiki/Go

https://www.youtube.com/watch?v=Z060uunNZ90

# 2.Visão geral

## 2.1 Arquitetura do programa

Cliente-servidor distribuído

## 2.2 Premissas de desenvolvimento

● O programa deve ser implementado em Python;

● O programa deve usar PyNetGames como suporte para execução distribuída;

● Além do código, deve ser produzida especificação de projeto baseada em UML, segunda versão.

# 3. Requisitos de Software

## 3.1 Requisitos funcionais

**RF001** – **Add Listener:** Trata do registro de uma subclasse de PyNetgamesServerListener no componente remoto do framework, uma instância de PyNetgamesServerProxy. É comum que esta subclasse de PyNetgamesServerListener seja também a classe que implementa a interface gráfica do jogo, dessa forma permitindo a atualização visual quando por exemplo uma jogada remota é recebida.

**RF002** – **Send Connect:** Trata da conexão com o componente remoto (o servidor) do framework. Diferentes execuções de um mesmo jogo precisam estar conectadas no mesmo servidor para que seja possível a disputa de partidas. Dessa forma, é preciso atentar-se ao endereço de servidor que será informado a PyNetgamesServerProxy. Caso nenhum endereço seja informado, será possível a comunicação somente na mesma máquina.

**RF003** – **Receive Connection:** Trata da confirmação de sucesso de uma conexão solicitada no caso de uso Send Connect.

**RF004** – **Send Match:** Trata da solicitação de uma partida para a quantidade de jogadores informada. Importante notar que cada jogo possui um identificador gerado pelo framework, em um arquivo gameid.txt. Diferentes execuções do mesmo jogo precisam ter o mesmo gameid.txt para que ocorra a conexão de partidas. Uma vez que o componente remoto do framework identifica que existem jogadores suficientes dada a quantidade solicitada, uma partida será iniciada (caso de uso Receive Match).

**RF005** – **Receive Match:** Trata do recebimento de uma partida, conforme solicitada no caso de uso Request Match. O parâmetro recebido é um objeto que contém os campos match\_id (que deve ser utilizado para enviar movimentos no caso de uso Send Move) e position, um inteiro que indica a vez do jogador.

**RF006** – **Send Move:** Trata do envio de um movimento para o componente remoto do framework. Importante destacar que o framework não realiza controle de “vez” dos jogadores, ficando isso a cargo da implementação do jogo com base na posição recebida no caso de uso Receive Match. Ao enviar o movimento, é necessário informar o id da partida, recebido no caso de uso Receive Match. O movimento enviado deve ser um dicionário composto por [tipos serializáveis](https://docs.python.org/3/library/json.html#json.JSONDecoder).

**RF007** – **Receive Move:** Trata do recebimento de uma jogada, O parâmetro recebido é um objeto que contém os campos match\_id, o identificador da partida, e payload, o dicionário contendo a jogada remota. O programa deve então prosseguir para processar a jogada e atualizar a tela do jogo.

**RF008** – **Send Disconnect:** Trata da solicitação de desconexão com o componente remoto do framework. Caso não haja conexão em vigor, nada ocorre. Caso a partida esteja em andamento, os demais jogadores também serão desconectados.

**RF009** – **Receive Disconnect:** Trata do recebimento de uma desconexão, seja ela solicitada (caso de uso Send Disconnect) ou recebida por desconexão de outros jogadores. O jogo deve ser resetado para um estado inicial.

**RF010** – **Receive Error:** Trata do recebimento de eventuais erros que ocorram no framework. O jogo deve ser resetado para um estado inicial.

**RF011** – **Iniciar Partida:** O programa deve pedir o nome do jogador e em seguida se conectar a outro buscando uma nova partida

**RF012 -** **Reiniciar partida:** A interface possui a opção de reiniciar a partida que retorna o programa ao estado inicial do tabuleiro. As peças são retiradas e caso os dois jogadores aceitem é iniciada uma nova partida com os mesmos jogadores.

**RF013** – **Colocar Peça:** Quando no turno do jogador ele deve ser capaz de selecionar uma posição livre e legal do tabuleiro para colocar uma peça da sua cor. O jogador deve ser notificado de um lance irregular e o programa deve novamente aguardar a primeira ação do jogador habilitado, caso o lance regular a ação deve ser enviada para o oponente e ser processada localmente.

**RF014** – **Passar a vez:** Quando no turno do jogador, caso julgue necessário ele deve ser capaz de passar sua vez. Quando os dois jogadores passam a vez consecutivamente o jogo deve acabar e um deles deve ser declarado o vencedor.

## 

## 

## 3.2 Requisitos Não funcionais

**RNF001** – Linguagem de programação: Python

**RNF002** – Interface: Tkinter

**RNF003** – Especificação do projeto: Baseado em UML 2, utilizando Visual Paradigm

**RNF004** – Interface do programa: Será produzida baseando-se no protótipo feito, como mostra a figura abaixo:

