

# Halma

## Especificação de Requisitos de Software

Beatriz Aguiar, Gustavo Fukushima, Margarida Vieira

August 2023

REQ\_PR\_halma\_2023aug25

Versão 1.1

25/09/2023

Versão	Autor(es)	Data	Ação
1.0	Beatriz Aguiar, Margarida Vieira e Gustavo Fukushima	07/09/23	Estabelecimento de Requisitos
1.1	Beatriz Aguiar, Margarida Vieira e Gustavo Fukushima	25/09/23	Ajustes após refinamento de casos de uso

# 1 Introdução

## 1.1 Objetivo do Desenvolvimento

Implementação de um programa para jogar Halma, possibilitando partidas entre dois jogadores. No jogo Halma, o objetivo é mover todas as peças até o canto oposto do tabuleiro através de movimentos estratégicos.

## 1.2 Definições e Abreviaturas

**RF** Requisito funcional

**RNF** Requisito não funcional

**RN** Regra de negócio

**Casa** Posição do tabuleiro

**Campo** Conjunto de casas de cada jogador

## 1.3 Referências

[Regras do Jogo](#)

# 2 Visão Geral

## 2.1 Arquitetura do Software

Cliente-servidor distribuído.

## 2.2 Premissas de Desenvolvimento

- O projeto deve ser produzido com *Visual Paradigm*.
- O programa deve ser implementado em *Python*.
- O programa deve utilizar a biblioteca *TKinter* para a construção da *interface* gráfica.
- O programa deve fazer uso da biblioteca *DOG* para suporte da execução distribuída.

## 3 Requisitos de Software

### 3.1 Regras de Negócio

**RN01 - Número de jogadores:** O jogo ocorre entre dois jogadores humanos.

**RN02 - Campo do jogador:** O campo de cada jogador é um conjunto de 19 casas. Os campos estão em cantos opostos do tabuleiro.

**RN03 - Movimentos permitidos:** Apenas são permitidos movimentos para posições ortogonais e diagonais.

**RN04 - Condição de saltos em jogada:** Saltos sobre peças são possíveis quando uma peça tanto do próprio jogador quanto do jogador adversário estiver adjacente a uma peça do jogador que está a executar o movimento.

**RN05 - Condição de vitória:** Ganha quem colocar primeiro todas as suas peças na área do oponente.

### 3.2 Requisitos Não Funcionais

**RNF01 - Linguagem de programação** O programa deve ser desenvolvido em *Python*.

**RNF02 - Arquitetura de software** O programa deve estabelecer uma conexão distribuída entre cliente e servidor através da ferramenta *DOG*.

**RNF03 - Menus** O programa deve possuir um menu inicial para iniciar a partida e inserir o nome do jogador.

**RNF04 - Interface gráfica** A *interface* deve ser implementada através da biblioteca *Tkinter*.

**RNF05 - Interface** A *interface* deve ser implementada de acordo com as imagens da secção 4.

### 3.3 Requisitos Funcionais

**RF01 - Início do programa** Ao entrar em execução, o programa deve solicitar o nome do jogador e permitir o início do jogo, realizando a conexão ao *DOG Server*, através de um botão.

**RF02 - Início do jogo** Após clicar no botão *Start*, e se estiverem reunidas as condições para se dar início ao jogo - a conexão ao *DOG Server* for bem sucedida e a condição mencionada em **RN01** -, o jogador deve ser notificado do início da partida e redirecionado para o ecrã de jogo, contendo o tabuleiro e o nome do jogador que deve realizar a jogada. O tabuleiro deve respeitar o indicado em **RN02**.

**RF03 - Seleção da peça a jogar** O programa deve permitir que o jogador, na sua vez, selecione uma das suas peças no tabuleiro através de um click. Essa funcionalidade só deve estar habilitada quando o jogador adversário, após terminar o seu turno, não tiver atingido a condição de vitória **RN05**.

**RF04 - Seleção do movimento a efetuar** A movimentação se inicia quando o jogador está habilitado para fazer sua jogada, **RF03**. O programa deve permitir que o jogador selecione uma posição de destino para a qual a peça deverá ser deslocada. Para que a movimentação seja válida, a casa de destino tem que se encontrar vazia e respeitar a **RN03**. Caso o movimento seja para uma casa adjacente o movimento encerra a jogada. Caso o movimento se trate de um salto, **RN04**, é necessário avaliar se existe a possibilidade de uma outra jogada consecutiva. A peça deve ter sua posição atualizada na *interface* gráfica, sendo transmitida essa informação para o servidor **RF07**.

**RF05 - Desistir do jogo** O programa deve conseguir finalizar uma partida quando um jogador clicar no botão *Exit*, na *interface* gráfica da aplicação. Assim, uma mensagem de abandono deverá ser enviada ao *DOG Server*.

**RF06 - Receber pedido de início de jogo** O programa deve conseguir receber e processar devidamente a notificação de início de jogo fornecida pelo *DOG Server*. Assim que receber a ordem dos jogadores por parte do mesmo, deve iniciar o procedimento suprarreferido em **RF02**, permitindo que o primeiro jogador execute o procedimento em **RF03**.

**RF07 - Receber informação de jogada** O programa deve conseguir receber as jogadas de um jogador ao final do respetivo turno. As informações do turno devem incluir, no mínimo, as posições inicial e final da peça que foi movida no turno. No caso de se tratar de uma jogada de salto, as informações do turno também devem incluir a peça do adversário sobre a qual a peça do jogador do turno saltou. A cada jogada, deve avaliar-se se a mesma gera condição de encerramento do jogo. Se sim, o jogo encerra com o nome do vencedor na tela, juntamente com um botão para executar o **RF02**. Caso contrário, o próximo jogador deverá poder começar o seu turno.

**RF08 - Receber desistência do oponente** O programa deve ser capaz de receber uma notificação de abandono de partida por parte do oponente remoto, enviada por *DOG Server*. Neste caso, a partida deve ser considerada encerrada e o abandono notificado na *interface* gráfica.

## 4 Capturas de Tela



Figure 1: Ecrã para inserir o nome

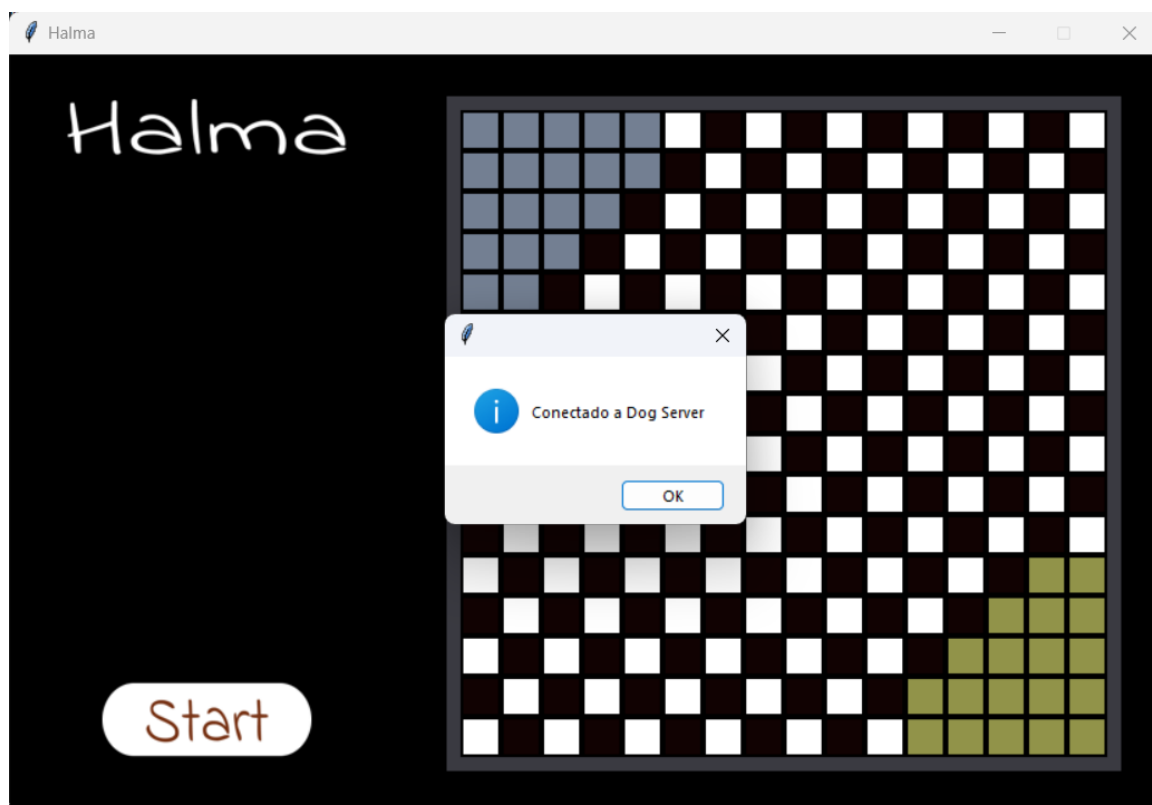


Figure 2: Ecrã com notificação de conexão ao servidor

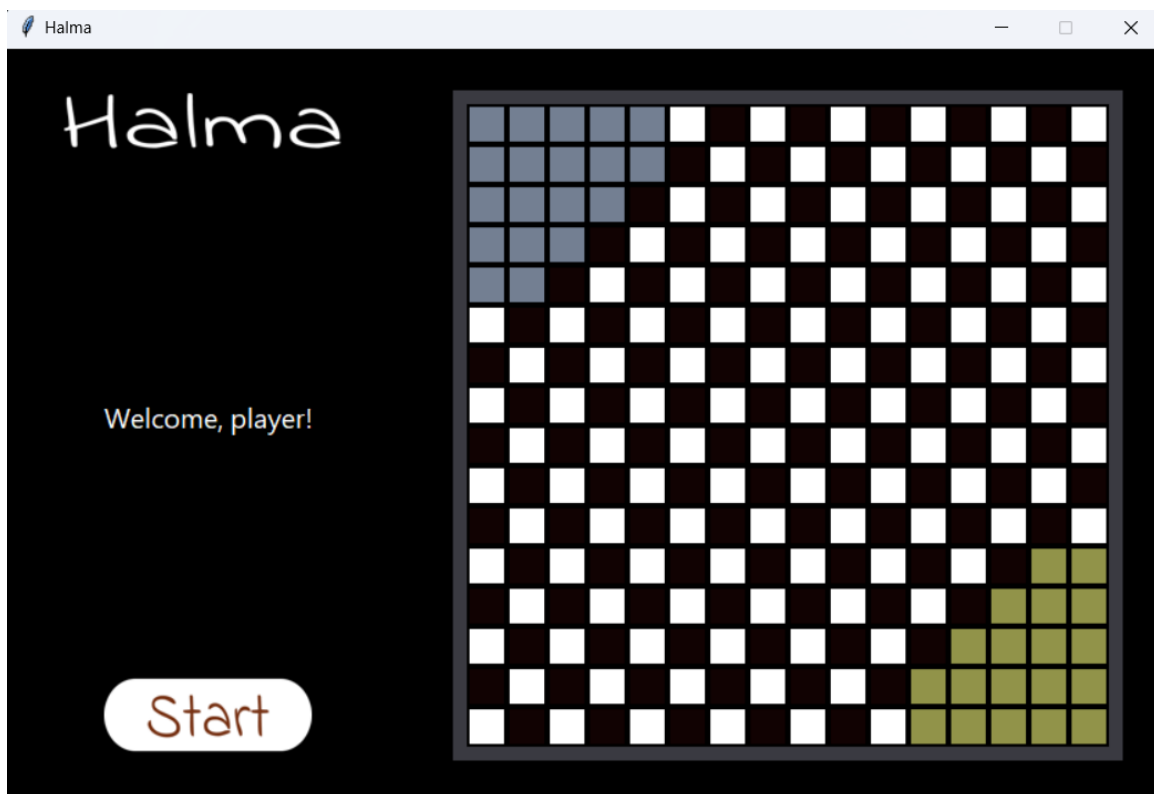


Figure 3: Ecrã inicial com nome do jogador





Figure 4: Jogada do primeiro jogador

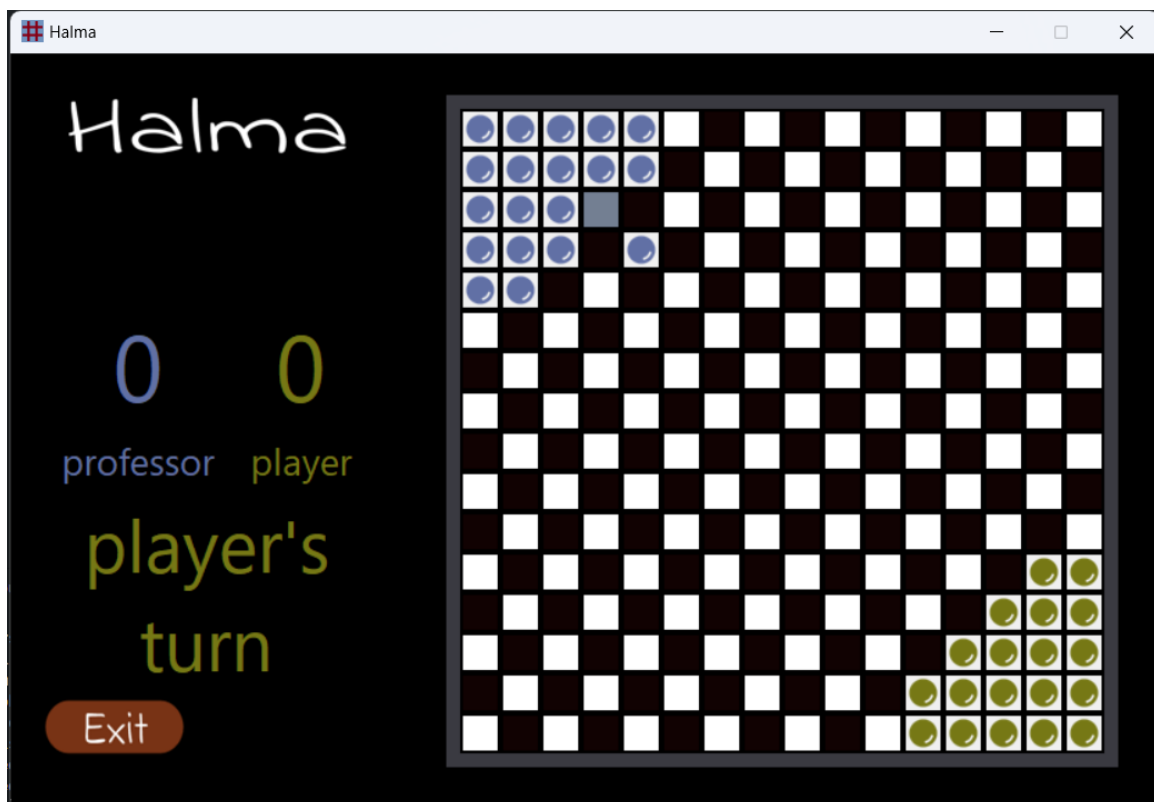


Figure 5: Jogada do segundo jogador