

# Especificação de Requisitos

## Archwizard Duel

Versão	Autores	Data	Ação
0.1	Alek Frohlich, Gabriel B. Sant'Anna e Mateus Favarin	10/09/2019	Eliciação inicial
0.2	Alek Frohlich, Gabriel B. Sant'Anna e Mateus Favarin	15/09/2019	Atualização da descrição do jogo e dos requisitos não funcionais
0.3	Alek Frohlich, Gabriel B. Sant'Anna e Mateus Favarin	25/11/2019	Atualização dos requisitos
0.4	Alek Frohlich, Gabriel B. Sant'Anna e Mateus Favarin	26/11/2019	Anexo das instruções de uso e descrição da linguagem interpretada

## Introdução

Em **Archwizard Duel**, cada jogador tem ao seu controle um mago com o qual deve provar seu valor ao vencer **duelos** contra seus oponentes. O combate acontece **em turnos**, cada um consistindo na simulação de um intervalo de tempo em uma arena. As ações dos magos são **programadas por código** escrito pelos jogadores em uma linguagem específica.

## Objetivos

Desenvolver um **jogo Player vs Player** (PVP) utilizando a ferramenta de jogos distribuídos **NetGames**. Este será entregue como [trabalho da disciplina INE5417](#) (Engenharia de Software I).

## Descrição do Jogo

- Uma partida consiste em uma arena com dois magos - cada um controlado por um jogador - os quais se alternam em turnos para lançar feitiços e se movimentar até que algum jogador vença.
- Cada ação de um mago gasta parte de um recurso seu chamado mana.
- Um turno consiste em uma sequência de ações efetuadas até que acabe a mana do mago controlado pelo jogador atuante (da vez) ou até que as ações propostas sejam esgotadas, momento em que passa a ser o turno do outro jogador, e assim por diante.
- Cada mago possui também um atributo de vida, que é reduzida quando ele é atingido por um feitiço inimigo.
- Uma partida termina quando a vida de um dos magos acaba, caracterizando vitória do jogador que controla o outro mago (que ainda tem vida).

- As ações realizadas pelos magos são controlados pelos seus respectivos jogadores através de jogadas.
- Uma jogada consiste em um código (texto) em uma linguagem de programação específica do jogo que define as ações a serem realizadas naquele turno (até que acabe a mana).
- Ações são, portanto, operações primitivas dessa linguagem que alteram o estado do jogo; fireball! (para lançar um feitiço) e step! (para movimentar o mago) são exemplos de tais operações.
- Essas primitivas, juntamente com [funções de primeira classe](#) e [estruturas de controle](#) (if-then-else, for, define, etc) compõem a linguagem de programação que serve de controle aos jogadores.
- A cada turno, o jogador deve escrever uma sequência de expressões válidas na linguagem, a qual será interpretada pelo programa - da mesma forma para ambos os jogadores - de maneira a trazer ao estado do jogo as alterações descritas nessa jogada.

**O Anexo 1 contém um tutorial de como jogar o jogo e uma listagem das construções sintáticas e operações primitivas disponíveis na linguagem de Archwizard Duel.**

## Referências

Jogos com ideias semelhantes e seus respectivos estilos de controles programáticos:

- [Gladiabots](#) (2019) - Programação visual através de fluxogramas.
- [Robocode](#) (2001) - Programação em código, semelhante a Java.
- [Crobots](#) (1985) - Programação em código, semelhante a C.
- [RobotWar](#) (1981) - Programação em código, semelhante a BASIC.
- [Color Robot Battle](#) (1981) - Programação em código, semelhante a Assembly.

## Visão Geral

### **Arquitetura do Programa**

Programa **orientado a objetos**, **distribuído** e **multiusuário** (dois jogadores).

### **Premissas de Desenvolvimento**

- A implementação deverá ser na linguagem Java.
  - Deve ser compatível com o [Java Runtime Environment \(JRE\) versão 8](#).
- Deve utilizar o framework [NetGamesNRT](#) para execução distribuída.
- Deverá ser entregue a modelagem do software em UML 2 produzida com a ferramenta [Visual Paradigm](#) (*Community Edition*)
- O programa deve apresentar uma interface gráfica bidimensional.

### **Interface Gráfica**

Segue abaixo um rascunho da interface gráfica que poderia ser apresentada pelo cliente do jogo. Destaca-se, entretanto, que são mostrados apenas elementos (botões) referentes ao

estado de uma partida em andamento. Estes seriam substituídos por opções como “Conectar à Sessão” e “Criar Sessão” ou “Iniciar Partida” e “Desconectar” dependendo do momento atual da interação do usuário com o programa (se está ou não conectado a uma sessão juntamente com outro usuário remoto, por exemplo).



## Requisitos de Software

### *Requisitos Funcionais*

- **Criar sessão:** O programa deve fornecer em sua interface a opção de estabelecer uma sessão de jogo para que outros usuários possam se conectar.
- **Conectar à sessão:** O programa deverá estabelecer a conexão entre os dois jogadores através do servidor NetGames. Cada jogador deve se identificar com um nome para seu personagem.

- **Desconectar da sessão:** O programa poderá desligar a conexão entre os jogadores caso estes não desejem iniciar novas partidas.
- **Iniciar partida:** O programa deve conter um botão que inicia a partida (caso já não exista alguma em andamento).
- **Desistir da partida:** O programa deve conter um botão para abandonar uma partida em andamento.
- **Enviar jogada:** O programa deve disponibilizar uma caixa de texto onde o jogador poderá digitar o código que servirá como esquema de controle do seu personagem em uma partida, com um botão que finaliza a jogada atual e a envia para ser avaliada.
- **Receber jogada:** O programa deverá mostrar também as jogadas realizadas pelo jogador remoto, atualizando o cliente do usuário local.
- **Receber determinação de início:** O programa deve avisar ao usuário que a partida foi iniciada, mostrando a interface apropriada.
- **Receber notificação de desconexão:** O programa deve atualizar a interface notificando o usuário em casos de perda de conexão com o servidor ou com o usuário remoto.

### ***Requisitos Não Funcionais***

- **Especificações de projeto:** O programa deverá ser escrito em Java, deverá ser compilado de forma a ser compatível com a JRE de versão 8 e deve seguir uma modelagem UML2 conformante com a metodologia vista em aula.
- **Framework distribuído:** O programa deve utilizar o framework NetGamesNRT para realizar a comunicação cliente-servidor.
- **Interface gráfica:** O programa deve possuir uma interface gráfica que represente o estado compartilhado do jogo. Esta interface será implementada em Java Swing por questões de compatibilidade.
- **Características da linguagem:** A linguagem do jogo deverá tomará uma forma semelhante à de [Lisp](#). Devendo prover primitivas para controlar o personagem (movimento, feitiços) e definir o fluxo de execução de suas ações (desvios condicionais, procedimentos).

## **Anexo 1 – Instruções de uso e descrição da linguagem**

Encontra-se a seguir o Anexo 1, que contém um breve tutorial sobre como jogar o jogo e a descrição da linguagem de programação utilizada em Archwizard Duel.

# Como jogar Archwizard Duel

## Configurando o servidor NetGames

Para que seja possível jogar o jogo, deve haver uma instância do servidor NetGames executando em um certo endereço de IP (podendo ser inclusive na máquina local, caso em que o IP a ser usado é *localhost* ou 127.0.0.1). Esse endereço deve ser informado por ambos os jogadores ao se conectarem na sessão.

## Conectando-se em uma sessão

- Um jogador deve ser o *host* da sessão, criando-a em um endereço de IP válido (onde haja um servidor NetGames) com o botão “*Create Session*”. O *host* deve ser o único usuário conectado no momento em que a sessão for criada.
- O outro jogador deve utilizar esse mesmo endereço de IP para entrar como convidado na sessão através do botão “*Join Session*”.

## Iniciando uma partida

- Na interface de sessão, o jogador que se conectou como cliente deve esperar o início da partida, ou deixar a sessão com “*Quit Session*”.
- Somente o *host* pode iniciar a partida, mas somente se houver algum outro jogador conectado naquela sessão. Basta então pressionar o botão “*Start Match*”.

## Jogando Archwizard Duel

Nesse ponto, o mapa do jogo é mostrado a ambos os jogadores, assim como os pontos de vida e magia do personagem associado ao usuário.

Existem na arena três tipos de elementos:

- **Rochas**, que são permanentemente fixas no mapa e podem ser usadas como barreiras contra feitiços.
- **Magos**: os personagens controláveis por cada jogador. **O *host* da sessão controla o mago que inicia o jogo no canto esquerdo do mapa, enquanto o outro usuário controla o que aparece no canto direito.**
- **Magias**, que são invocadas pelos magos e ao atingir um inimigo reduzem seus pontos de vida.

O ganhador de um duelo é aquele que conseguir fazer com que o mago adversário perca todos os seus pontos de vida.

Cada jogada é realizada seguindo os comandos programados pelo jogador e enviados em seu turno. **Um turno termina quando todos os comandos forem executados ou quando houver algum problema na execução do código** (inclui tentar executar uma ação sem ter mana suficiente, acessar uma variável indefinida, somar elementos que não são números, entre outros).

A linguagem se aproxima de LISP e conta com alguns mecanismos para controle de fluxo de execução, operações aritméticas, definições de variáveis, criação de procedimento e utilização de funções de primeira ordem. Segue abaixo um exemplo didático:

```
; todo texto que vem depois de um ';'
; é tratado como comentário

; definindo uma variavel x com o valor `false`
(define x false)

; aqui definimos um procedimento `foo!` que recebe 2 parametros:
; uma outra funcao sem argumentos e uma direcao
(define foo! (lambda (test dir)
  ; abaixo um uso da condicional if-then-else
  (if (test)           ; se a funcao (test) retornar algo verdadeiro...
      (begin           ; um bloco begin é executado sequencialmente
        (step!)        ; o mago da um passo à frente
        (fireball!))   ; e lança uma bola de fogo naquela direcao
        (turn! dir)))  ; caso contrário, ele se vira na direção dada

; por fim, podemos invocar `foo!` passando os parametros a seguir
(foo!
  ; o 1o argumento é uma funcao que verifica se o caminho à frente está livre
  (lambda () (not (blocked?)))
  ; e o segundo é a direcao para qual o mago pode virar
  RIGHT)

; algumas operacoes aritmeticas basicas tambem estao presentes
(+ 1 1) ; resulta em 2, como esperado
```

## Uma API mágica

Seguem listados a seguir os procedimentos e valores primitivos disponíveis na linguagem de Archwizard Duel.

- UP: valor utilizado para representar a direção de movimentação acima.
- DOWN: valor utilizado para representar a direção de movimentação abaixo.
- LEFT: valor utilizado para representar a direção de movimentação à esquerda.
- RIGHT: valor utilizado para representar a direção de movimentação à direita.
- (turn! dir): vira o mago para a direção desejada, devendo ser alguma dentre UP, DOWN, LEFT ou RIGHT. Custa 5 de mana.
- (step!): faz com que o mago dê um passo à frente, consumindo 10 de

mana.

- `(fireball!)`: lança uma magia de bola de fogo na direção à qual o mago está virado. O feitiço é dissipado após atingir algum obstáculo e causa 20 pontos de dano ao oponente se tocá-lo. Custa 30 pontos de mana.
- `(blocked?)`: confere se o mago pode dar um passo à frente, ou se o caminho está bloqueado.
- `(mana)`: retorna a mana disponível nesse instante.
- `(health)`: retorna a vida disponível nesse instante.
- `false`: valor booleano falso.
- `true`: valor booleano verdadeiro.
- `(not x)`: inversão lógica do argumento `x`.
- `(= a b)`: comparação de igualdade entre valores numéricos.
- `(< a b)`: comparação `a` menor que `b`.
- `(> a b)`: comparação `a` maior que `b`.
- `(<= a b)`: comparação `a` menor ou igual que `b`.
- `(>= a b)`: comparação `a` maior ou igual que `b`.
- `(+ a b)`: soma `a` e `b`.
- `(- a b)`: subtrai `b` de `a`.
- `(* a b)`: multiplica `a` e `b`.
- `(/ a b)`: divide `a` por `b`, gera um erro quando `b` é zero.
- `(round x)`: retorna o valor `x` arredondado ao inteiro mais próximo.
- `(quotient a b)`: resultado da divisão inteira de `a` por `b`, gera um erro quando `b` é zero.
- `(remainder a b)`: resto da divisão inteira de `a` por `b`, gera um erro quando `b` é zero.
- `(boolean? var)`: confere se `var` é um valor booleano.
- `(number? var)`: confere se `var` é um valor numérico.