Projeto Rummikub Twist

Especificação de Requisitos de Software

Versão 1.1

07/12/2024

Versão	Autores	Data	Ação
1.0	Tom Pereira Hunt, Lucas de Souza Vieira, Gabriel Salmoria	18/09/2024	Estabelecimento dos requisitos
1.1	Tom Pereira Hunt, Lucas de Souza Vieira, Gabriel Salmoria	07/12/2024	Mudança de alguns requisitos e regras que mudaram ao longo do desenvolvimento dos diagramas e do código.

Conteúdo:

1 - Introdução

1.1 - Objetivo

Criação de um software distribuído que permita e organize jogos de Rummikub Twist na modalidade jogador contra jogador.

1.2 – Definições e abreviaturas

Regras do jogo: ver apêndice

1.3 – Referências:

Apresentação das regras do jogo (vídeo do canal Vem Ka Jogar):
https://www.youtube.com/watch?v=lSvzz6odV7A

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 DOG 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

Requisito funcional 1 – Iniciar programa: ao ser executado, o programa deve apresentar na interface a tela de início, que contém os botões de fechar o jogo, iniciar a partida, bem como uma caixa de texto para inserir o nome do jogador. Em seguida, usa os recursos do DOG para se conectar ao servidor DOG. O jogo deverá mostrar para o jogador se houve (ou não) sucesso na conexão ao servidor. Somente quando a conexão for estabelecida com sucesso (e o usuário avisado que isso ocorreu), que os outros botões serão liberados. Caso a conexão não seja possível, o único botão que deve ser ativado é o de encerrar o jogo.

Requisito funcional 2 – Iniciar jogo: O programa precisa possuir um botão de "iniciar partida" para o início de uma nova partida. Este botão só será liberado quando o usuário digitar um nome na caixa de texto, que pede para o usuário inserir seu nome. O processo de início de jogo envolve o envio de uma solicitação ao Dog Server, que responderá com o resultado: em caso de sucesso, será fornecida a identificação e a ordem dos jogadores; caso contrário, será informada a razão pela qual o jogo não pode ser iniciado. Em seguida, a interface do programa será atualizada de acordo com a resposta do Dog Server – Isto é, a partida inicia, e o jogo inicia, com as peças de cada jogador disponíveis apenas para eles mesmos. A ordem de jogada dos jogadores deverá ser decidida de forma aleatória no início de toda partida. A interface deverá estar habilitada para o procedimento de jogada do primeiro jogador.

Requisito funcional 3 – Movimentar peça: O programa deverá permitir que o jogador que estiver com a vez ativa possa arrastar uma peça presente em sua área de peças, ou então outras peças da "mesa" (a partir da segunda jogada do jogador). A peça sendo arrastada deve ser visualmente destacada na interface do programa, tanto durante o movimento, quanto após, pelo resto da jogada. Caso a peça movimentada já estava na mesa, o lugar original da peça deve sofrer uma animação simples para indicar que foi movida de lá.

Uma jogada será considerada "não permitida" se o jogador não movimentar a peça dentro dos espaços possíveis. Isto é, se o jogador tentar movimentar uma peça da mesa para outra área da interface, a peça imediatamente volta para onde estava. A única exceção é se a peça foi colocada na mesa naquela mesma rodada, e o jogador está tentando botar ela de volta na sua área de peças. Além disso, caso o jogador tente movimentar uma peça da sua área de peças para um lugar fora da mesa (a área de jogo), a peça também deve voltar para onde estava. Depois de cada um desses movimentos "não permitidos", o programa deve voltar a esperar por um movimento do jogador.

No caso de um lance ser considerado "permitido", o programa deve enviar a jogada ao(s) adversário(s) por meio dos recursos do DOG. O programa também deve verificar se ao final de um movimento, a partida se encerrou, com base nas regras do jogo (ver apêndice). No caso de encerramento de partida, deve ser notificado o nome do jogador vencedor; no caso de não encerramento, o jogador pode continuar fazendo movimentos (se ainda possuir tempo para a jogada; ver mais em requisito funcional 7 – Limitar o tempo por jogada).

Requisito funcional 4 – Passar a vez: O programa deve apresentar um botão de passar a vez na interface, o que permite que um jogador alterne a vez de jogada para o próximo jogador. O jogador poderá apertar este botão a qualquer momento *durante a sua jogada*, mas com consequências diferentes, dependendo de suas ações no jogo:

1) caso o jogador tenha feito pelo menos um movimento válido, completando pelo menos uma sequência, sem deixar peças "soltas" (especificação das regras de jogo – ver apêndice), este botão passa a vez para o próximo jogador.

2) caso o jogador não tenha feito nenhum movimento válido, o programa simplesmente compra uma peça para o jogador, e passa a vez para o próximo jogador (ver requisito funcional 5 – comprar peça).

3) caso o jogador tenha feito movimentos válidos, mas ainda há peças "soltas", isto é, não fazendo parte de uma sequência válida, o programa irá reverter todos os movimentos feitos pelo jogador naquela rodada, e deverá comprar uma peça pra ele.

Requisito funcional 5 – Comprar peça: O programa deve permitir que os jogadores realizem, durante suas jogadas, a ação de comprar uma peça. Isto é, permitir que um jogador, caso não possua nenhum movimento válido, ou seja parte de sua estratégia, realizar a ação de compra.

O programa permitirá isso por meio o botão de passar a vez (ver mais em requisito funcional 6), que deve ser um elemento da interface, e com que cada jogador possa interagir durante sua jogada. Quando este último for clicado, o programa deve adicionar uma nova peça na área de peças do jogador que clicou neste elemento, e passa a vez para o próximo jogador da ordem.

Requisito funcional 6 – Limitar o tempo por jogada: O programa deve apresentar, em sua interface gráfica, um *timer*, que não será interativo (isto é, os jogadores não poderão interagir diretamente com ele). Esse timer deverá fazer uma contagem regressiva do tempo máximo para

cada jogada. Além disso, ele deve gerar situações diferentes dependendo das decisões dos jogadores:

1) Se o jogador clicar no botão “passar a vez” antes do tempo acabar, isso deve fazer com que o timer se reinicie, e comece a contar de novo a partir do momento que a próxima rodada começar.

2) Se o jogador deixar o tempo do timer se esgotar, ou seja, chegar à marca de 0 segundos sobrando, a jogada dele é encerrada, e:

- Caso ele tenha feito algum(s) movimento(s) válido(s) e não deixado nenhuma sequência incompleta, o programa deve simplesmente passar a vez para o próximo jogador, e reiniciar o timer.
- Caso ele tenha feito algum(s) movimento(s) válido(s), mas deixado alguma sequência incompleta, todos seus movimentos devem ser reiniciados (até os que completaram alguma sequência), e uma carta é comprada pra ele automaticamente. Além disso, o programa deve reiniciar o timer, e passar a vez para o próximo jogador.
- Caso o jogador não tenha feito nenhum movimento permitido (ou seja, a mesa e sua área de peças estão iguais ao que estavam no início da rodada), uma peça deverá ser comprada automaticamente para ele. Em seguida, o programa deve reiniciar o timer, e passar a vez para o próximo jogador.

Requisito funcional 7 – Receber determinação de início: O programa deve ser capaz de receber uma notificação de início de jogo enviada pelo Dog Server, em resposta a uma solicitação de início feita por outro jogador conectado ao servidor. Após o recebimento dessa notificação, o procedimento segue o mesmo descrito no ‘Requisito funcional 2 – Iniciar jogo’, ou seja, a interface do programa deve ser atualizada com as informações recebidas, e, se o jogador local for o responsável por iniciar a partida, a interface deve permitir que ele faça seu primeiro lance.

Requisito funcional 8 – Receber jogada/movimento: O programa deve ser capaz de receber a jogada do adversário, enviada pelo Dog Server, quando for a vez dele. A jogada recebida deve ser válida e conter as informações previstas no ‘Requisito Funcional 3 – Movimentar Peça’. O programa deve remover a peça da posição de origem e colocá-la no destino.

Em seguida, deve verificar se a partida foi encerrada. Isso deverá ser feito pelo programa com base nas regras do jogo (ver apêndice). Caso o jogo tenha terminado, o nome do jogador vencedor deve ser informado; caso contrário, o jogador local deve ser habilitado para realizar seu lance.

Requisito funcional 9 – Receber notificação de abandono: o programa deve ser capaz de receber uma notificação desistência da partida por parte do adversário remoto, enviada pelo Dog Server. Nesse caso, a partida deve ser considerada finalizada, e a desistência deve ser informada na interface.

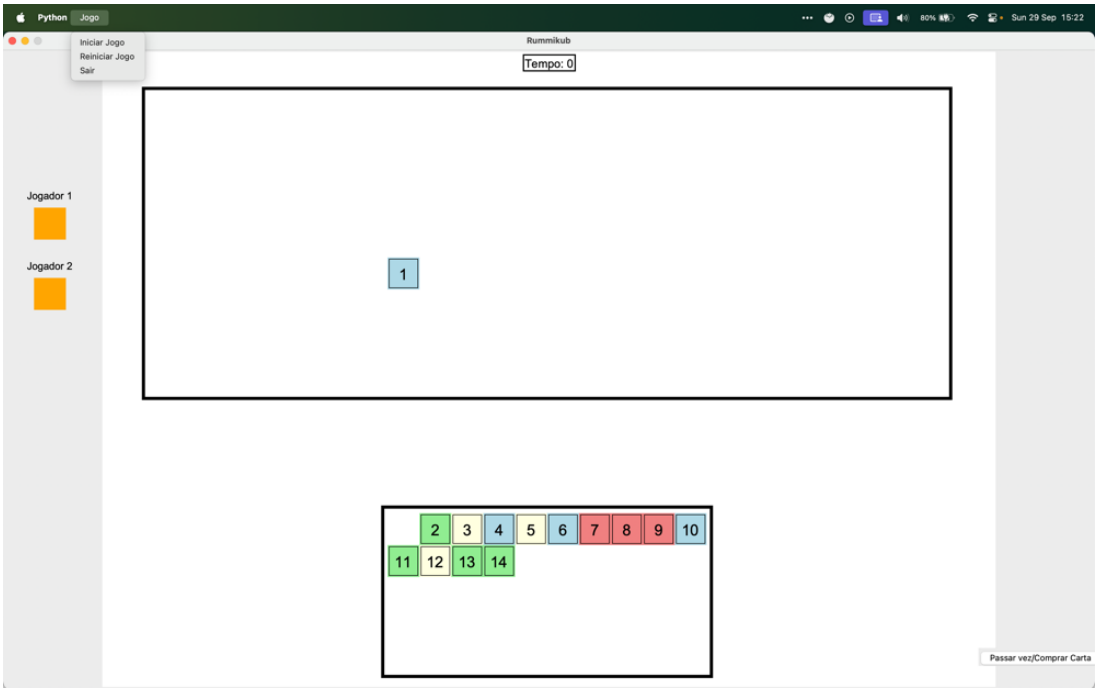
3.2 – Requisitos Não Funcionais

Requisito não funcional 1 – Linguagem da implementação: O programa deve ser implementado na linguagem Python 3;

Requisito não funcional 2 – Tecnologia de interface gráfica para usuário: A interface gráfica deve ser baseada em *TKinter*;

Requisito não funcional 3 – Suporte para a especificação de projeto: A especificação de projeto deve ser produzida com a ferramenta Visual Paradigm;

Requisito não funcional 4 – Interface do Programa: A interface do programa será produzida conforme o esboço da imagem abaixo.



Apêndice: Regras de Rummikub Twist

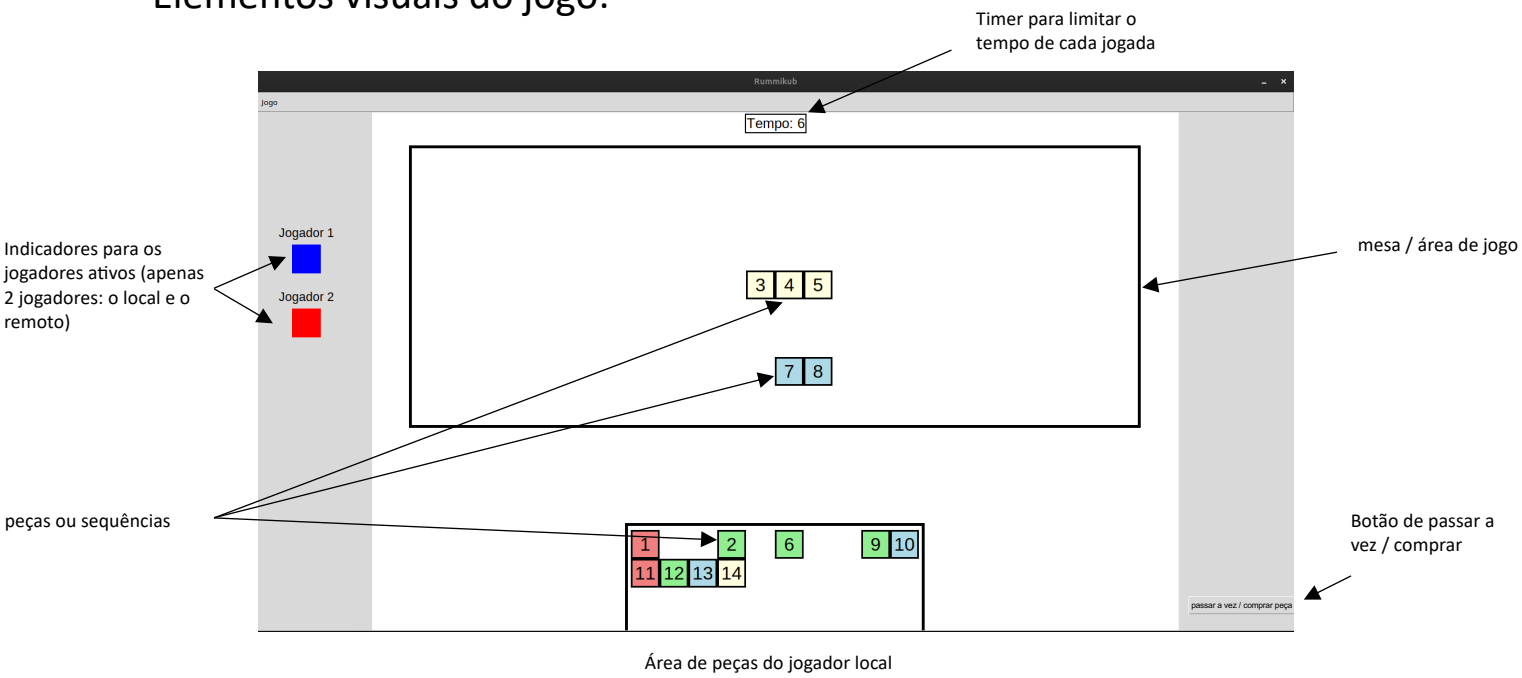
O objetivo do **Rummikub Twist** é ser o primeiro jogador a se livrar de todas as suas peças, criando combinações válidas no tabuleiro.

Componentes do Jogo:

- 104 peças numeradas de 1 a 13, em quatro cores diferentes (vermelho, azul, amarelo, preto), com duas cópias de cada número.

- 8 coringas (2 de cada tipo: coringa clássico, coringa duplo, coringa espelho, coringa muda-cor). Cada coringa será representado por uma peça com um inteiro negativo. O coringa clássico será representado por -1, o coringa duplo pelo -2, o coringa espelho pelo -3, e o coringa muda-cor pelo -4.

Elementos visuais do jogo:



Regras Gerais:

1. **Número de Jogadores**: 2

2. **Distribuição das Peças**: Cada jogador começa com 14 peças. O restante fica no "bolo", de onde os jogadores puxam se necessário.

3. **Jogadas**: O jogador precisa colocar no tabuleiro uma combinação de peças. As combinações podem ser:
    - **Sequência**: Três ou mais peças do mesmo número em cores diferentes.
    - **Grupo**: Três ou mais peças em ordem numérica consecutiva da mesma cor.

4. **Turno dos Jogadores**: A cada turno, o jogador pode:
    - Colocar uma ou mais peças na mesa formando uma combinação válida.

- Puxar uma peça do bolo se não conseguir ou preferir não jogar.
- Usar coringas para completar ou alterar combinações já existentes.

5. **Regras de Combinações**:
   - **Grupo**: Deve ter no mínimo três peças com o mesmo número, mas de cores diferentes.
   - **Sequência**: Deve ser composta por pelo menos três peças consecutivas da mesma cor.

6. **Uso de Coringas**:
   - **Coringa Clássico**: Substitui qualquer peça em uma sequência ou grupo.
   - **Coringa Duplo**: Funciona como duas peças consecutivas em uma sequência ou como dois números iguais em um grupo. Por exemplo, permita a sequência "7, 8, coringa duplo, 11, 12", com o coringa duplo valendo por "9, 10".
   - **Coringa Espelho**: Espelha qualquer combinação de peças no jogo. Por exemplo, se uma sequência for "2, 3, 4", o coringa poderia ser aplicado no 4, fazendo com que a sequência fique funcionalmente igual a "2, 3, 4, 4, 3, 2", sendo esse "4, 3, 2" representado pelo coringa.
   - **Coringa Muda-Cor**: Pode alterar a cor de uma sequência ou grupo. Por exemplo, se uma sequência for “3, 4, 5 vermelho”, o coringa pode mudar essas peças para outra cor. Assim, isso permite que seja colocado uma peça 6 azul, no lado direito, por exemplo.

7. **Manipulação do Tabuleiro**: Uma vez que o jogador fez sua primeira jogada, ele pode, em turnos subsequentes, reorganizar as peças no tabuleiro para formar novas combinações, desde que todas as combinações resultantes sejam válidas.

8. **Substituição de Coringas**: Um coringa pode ser substituído por uma peça apropriada em qualquer momento do jogo, desde que a combinação em que ele está inserido continue válida.

9. **Encerramento do Jogo**: O jogo termina quando um jogador se livrar de todas as suas peças, ganhando assim a partida.