Ludo - Linguagem de Programação I

Este documento descreve os requisitos e apresenta uma visão geral do trabalho da Disciplina de Linguagem de Programação I, sob orientação do professor Alan Moraes da Universidade Federal da Paraíba.

Documento desenvolvido como atividade da monitoria de Victor José de Sousa Koehler.

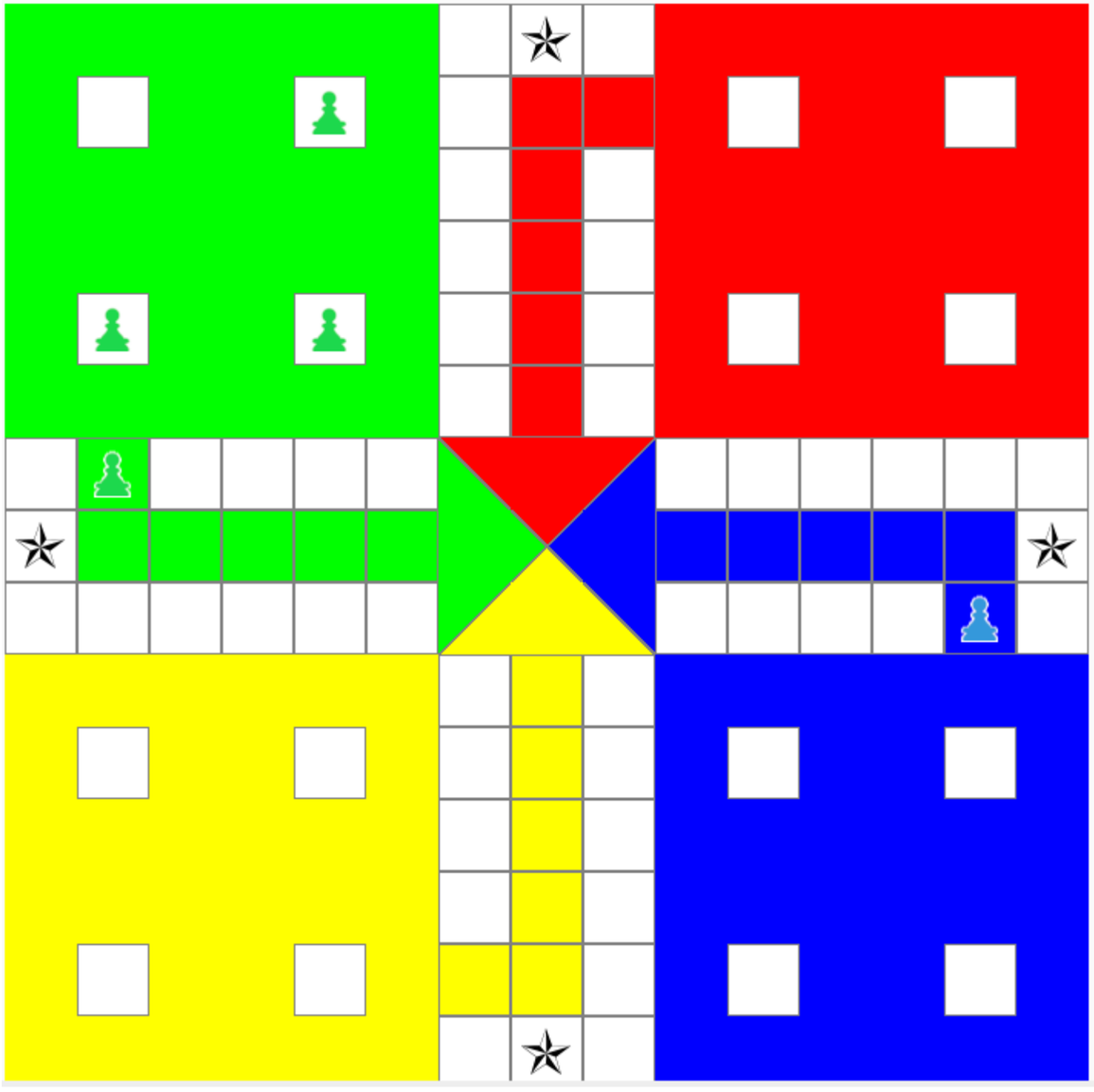


Fig. 1: Tabuleiro convencional de Ludo

Especificações do Trabalho

Introdução

O jogo é constituído de 2 a 4 jogadores, cada um possuindo 4 peões/peças, e o objetivo é cada um dos peões para o centro do tabuleiro dando uma volta completa partindo do ponto de início de cada jogador/cor:

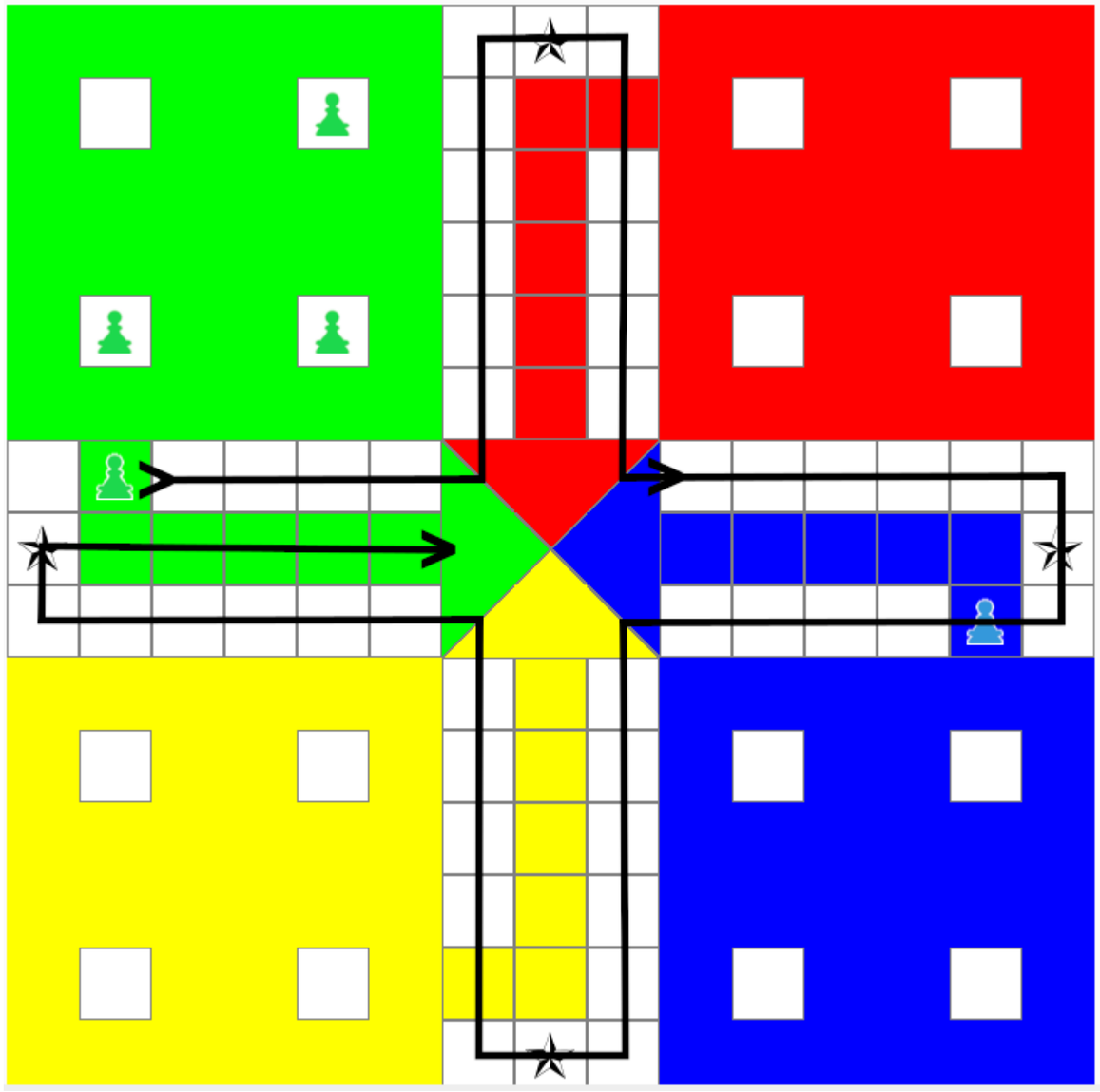


Fig. 2: Direção de movimento das peças.

Considerações Iniciais

Este pacote contém o "framework" na qual o trabalho deve ser implementado. Toda a lógica da interface gráfica já se encontra pronta e funcionando, cabendo aos discentes implementar as regras do jogo Ludo propriamente dito. Isto é, ao executar o projeto, os alunos encontrarão a seguinte tela:

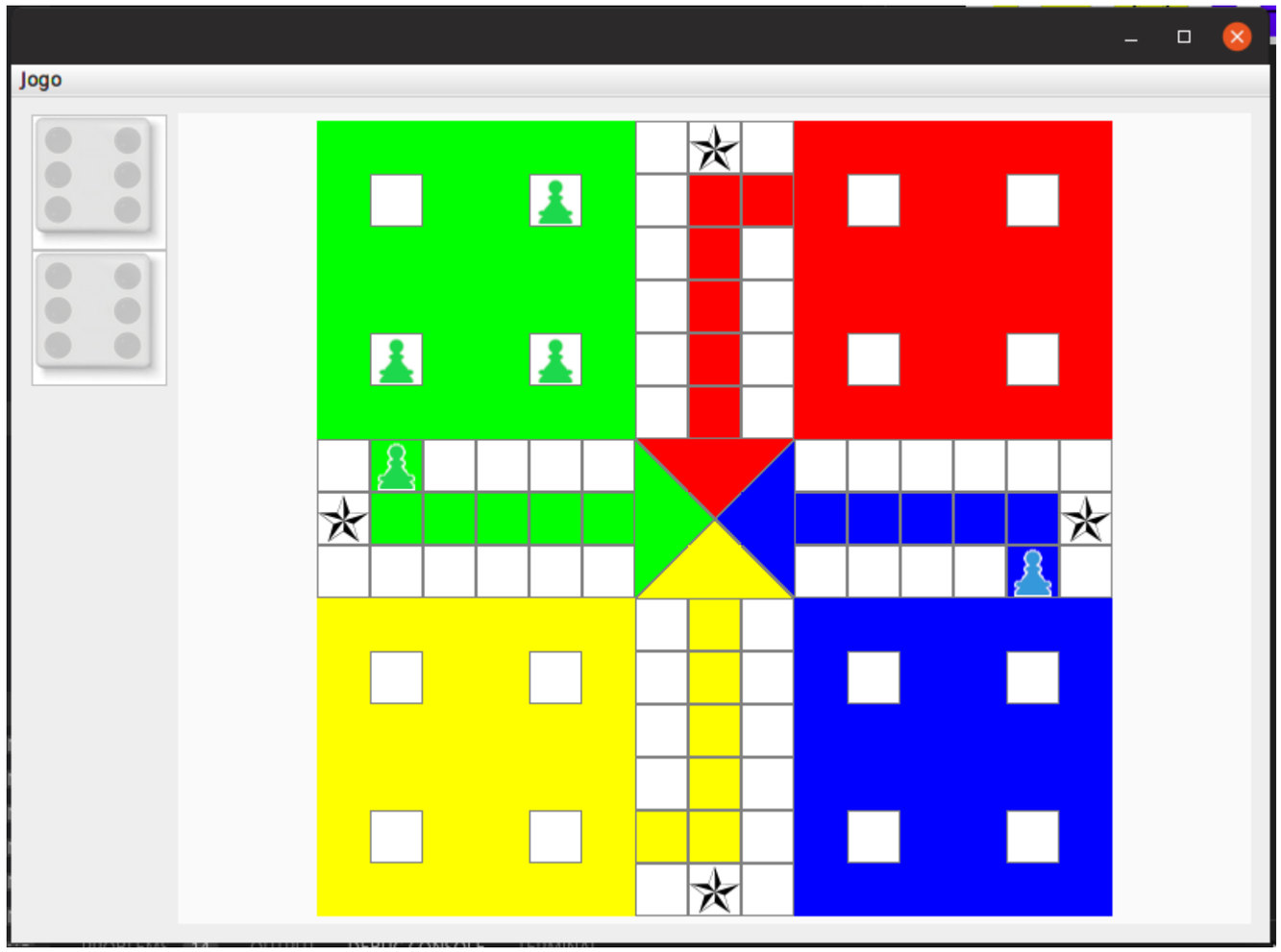


Fig. 3: Janela da aplicação.

Como pode ser percebido ao interagir com esta janela, não há regras em funcionamento: ao clicar nos dados eles serão arbitrariamente alterados e ao clicar em alguma peça, ela se moverá de acordo com os dados.

A imagem abaixo contém as terminologias usadas neste documento e em todo o código-fonte escrito:

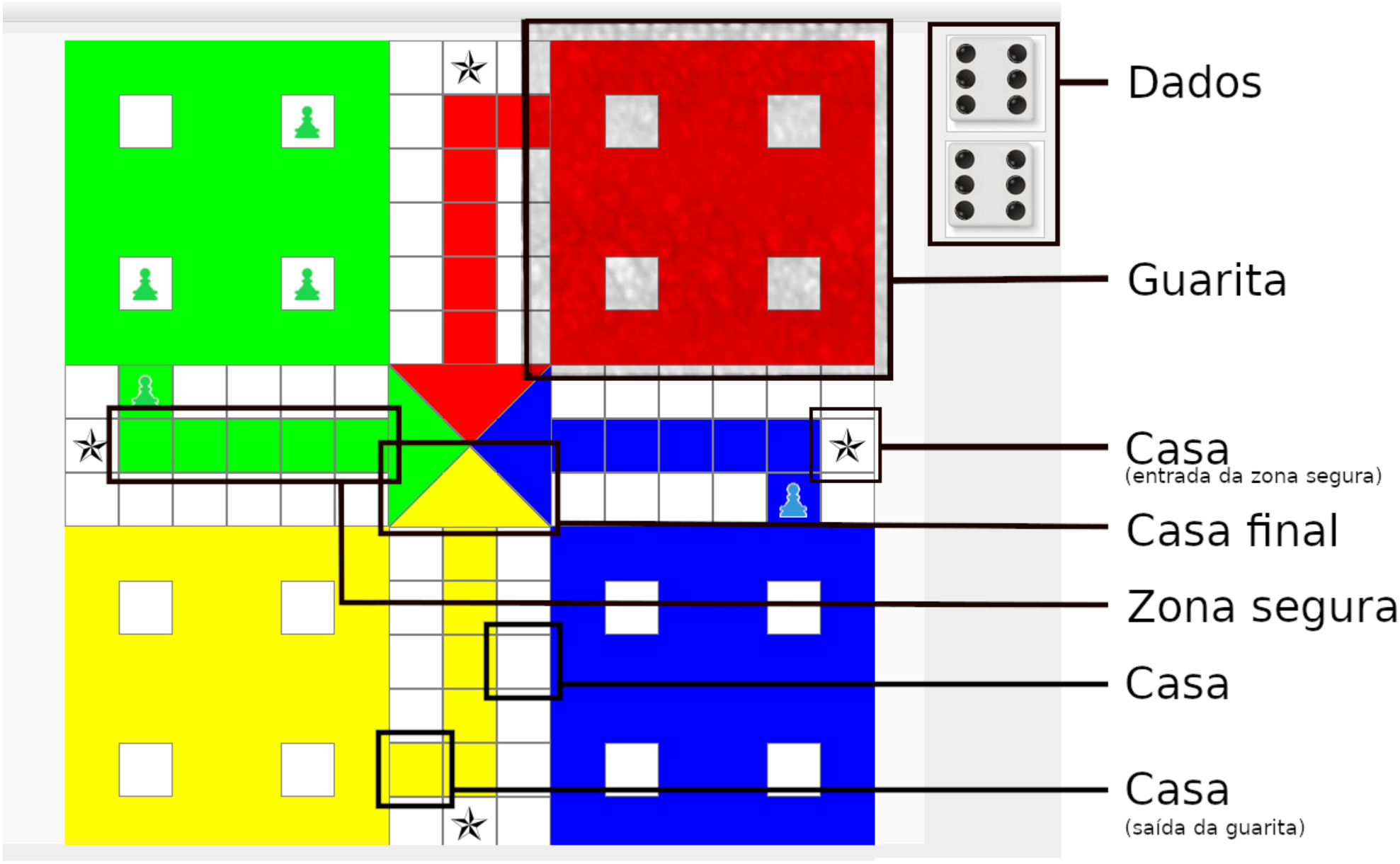


Fig. 4: Terminologias do projeto.

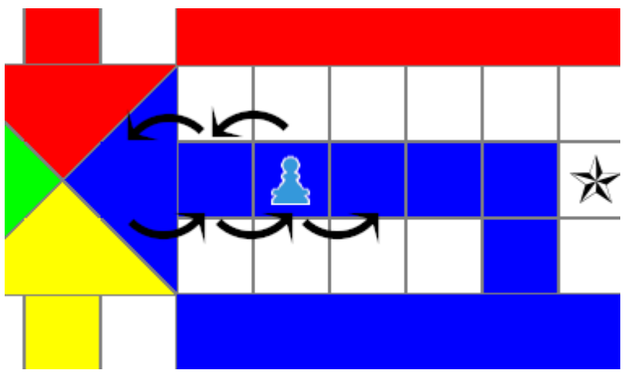
Regras

Esta é uma implementação simples de Ludo. As regras estão listadas a seguir de maneira objetiva. Caso a leitura se demonstre cansativa, sugere-se que o leitor tente buscar na Internet alguma fonte mais simples como base para ser introduzido ao jogo. Adverte-se, no entanto, que devido ao grande número de variações e particularidades do jogo presente mundo afora, as regras a seguir terão precedência sobre quaisquer outras. Interpretações próprias e modificações nas regras devem ser discutidas com o docente:

- Cada jogador começa com 4 peças na sua guarita.

- Os jogadores se revezam em turnos para jogar os dados.
- Toda vez que um jogador joga os dados, ele é obrigado a fazer algum movimento com os dados que obteve (isto é, ele não pode deixar perder a vez de propósito)
- Ao jogar os dados, soma-se os seus valores. Depois o jogador pode escolher uma peça no tabuleiro (fora da guarita) para se mover exatamente X casas, onde X é a soma dos dados, seguindo a direção horária (ver Fig. 2). Caso o jogador não tenha peças no tabuleiro, perde a vez.
- **Entretanto**, quando um jogador obtém dados iguais (isto é, quando ele tira 6-6, 5-5, etc), ele possui duas opções:
    i. Escolhe uma peça para sair da guarita, colocando-a na casa de início de sua cor (na imagem de referência, "**Casa (saída da guarita)**"). Após tirar a peça da guarita e colocá-la no jogo, deve jogar os dados novamente para decidir quantas casas a dita peça irá se mover (partindo do início, é claro).
    ii. Escolhe uma peça qualquer no tabuleiro (fora da guarita) para movê-la normalmente. Após isso, o jogador terá outra chance para lançar os dados (isto é, ele ganha o direito de jogar duas vezes). Se, por acaso, ele tirar novamente dados iguais, repete-se esse processo.
- Como consequência do passo anterior, se todos os peões do jogador estiverem na guarita, ele precisará tirar dados iguais para conseguir tirar um de lá. Senão, perde a vez. Isso também se aplica no início do jogo, onde todas as peças estão na guarita.
- Se, após o lançamento dos dados e escolha de uma peça **P** para se mover, houver uma peça adversária **Q** na casa de destino onde **P** irá se inserir, então **P** irá "comer" **Q**: ela volta para a guarita de sua respectiva cor, e **P** assume o lugar anterior dela.
- Não é permitida a uma peça "comer" outra do mesmo jogador.
- Não existe o conceito de torres, empilhamento ou outras variações: Uma casa pode possuir no máximo uma peça em cima dela. (Exceto a casa final, a qual será discutida mais a frente)
- Ao dar *(quase)* uma volta completa no tabuleiro, o peão que chega a entrada da casa segura (uma estrela, ver Fig. 4) continuará seu caminho pela "zona segura", isto é, pelas casas que possuem a mesma cor do peão.
- Um peão só pode entrar em uma "zona segura" de sua mesma cor. (Por isso o nome zona segura: adversários não podem comer as peças de um jogador dentro da zona).
- As casas da zona segura possuem a característica de ir e voltar:

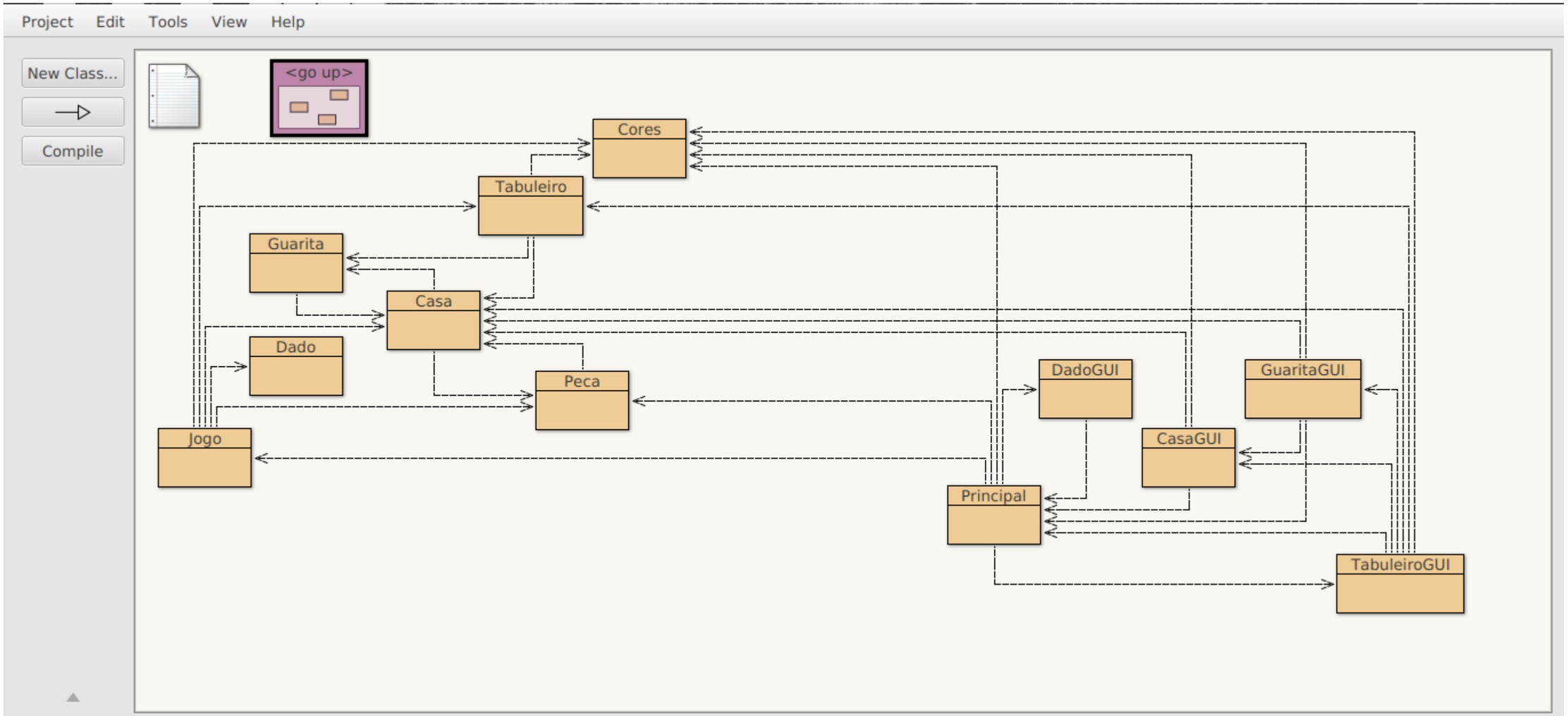
  Por exemplo, ao tirar 5 na soma dos dados, uma peça que recém entrou na zona segura vai sobrepassar a casa final e vai começar a "voltar". **Este estado não persiste entre turnos:** na próxima vez que o jogador escolher esta peça para jogar, ela se moverá em direção a casa final normalmente (isto é, o jogador precisará tentar achar a soma certeira nos dados para levar esta peça a casa final).



- Cumprindo o objetivo de levar todas as peças a casa final (que por sua vez, "permite" o "empilhamento" de várias peças), o jogador vence o jogo (e os demais jogadores).

O Framework

Os alunos devem fazer as modificações que se fizerem necessárias para implementar as regras do jogo somente nas classes que tratam do modelo do jogo diretamente e **não** possuem implementações de GUI, o que corresponde as classes mais a esquerda na imagem abaixo.



Neste trabalho, é fortemente desencorajado fazer mudanças nas classes que tratam da GUI (a direita na imagem), principalmente se o objetivo das alterações for implementar uma das regras básicas descritas acima.

No entanto, não é proibido fazer alterações estéticas ou melhorias simples (por exemplo, trocar as cores ou imagens) na GUI se o aluno desejar, embora isso não implique, necessariamente, que o trabalho ganhará pontos extras (Consultar o professor). Recomenda-se, nesses casos, que o aluno dê prioridade em terminar o mínimo exigido nas regras do trabalho e sempre mantenha backups das

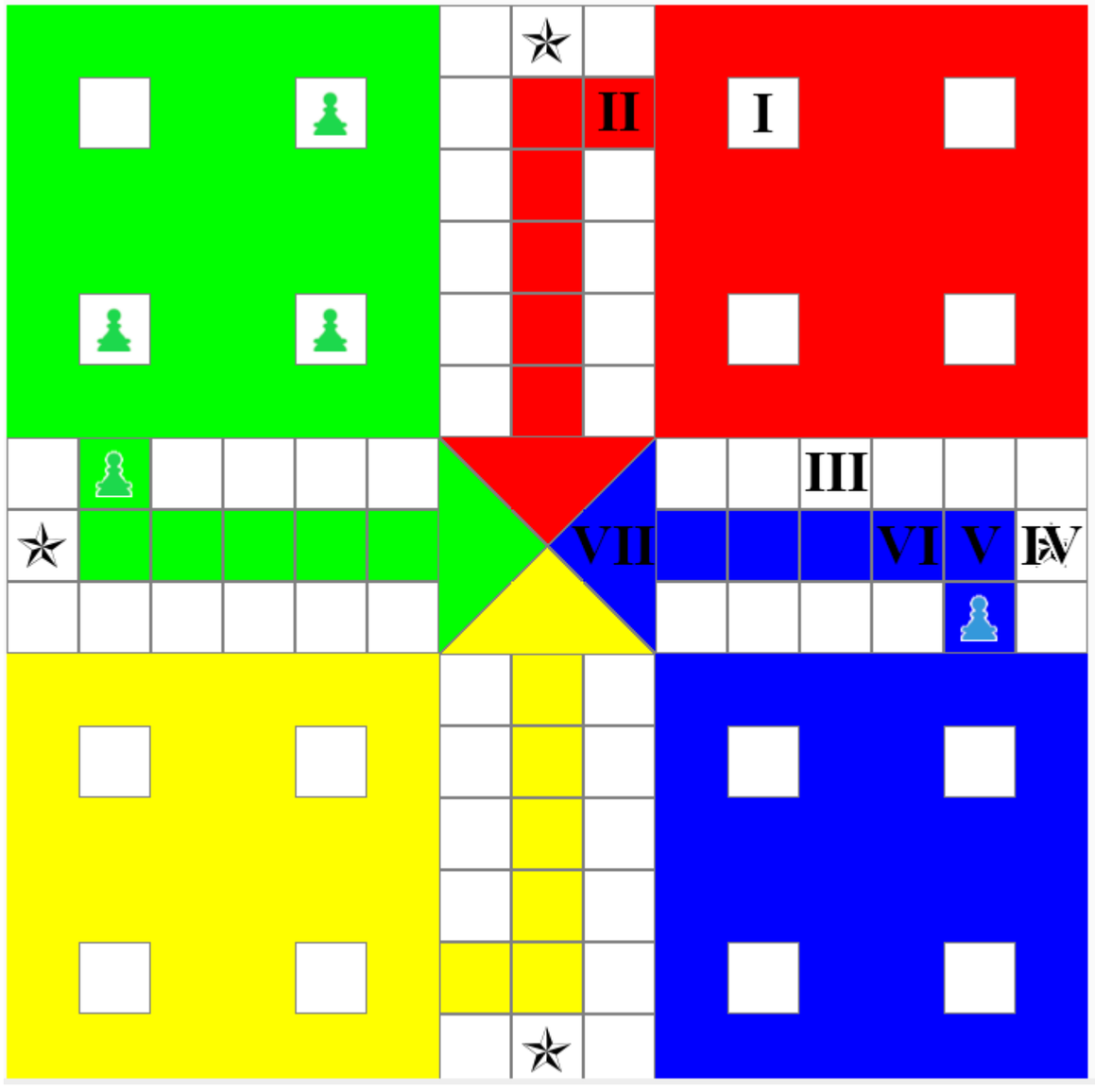
diferentes versões.

Documentação

O Javadoc gerado a partir do código-fonte pode ser encontrado aqui: javadoc/index.html

Nestes documentos, sugere-se a leitura, de **pelo menos**, *Casa* e *Jogo*. Especificamente em Jogo, encontra-se os dois únicos métodos modificadores que a GUI usa para controlar o jogo: `jogarDados()` e `escolherCasa(Casa casaEscolhida)` (Obviamente, a GUI usa de vários métodos de diferentes classes para desenhar na tela, mas somente esses dois "modificam" o estado do jogo. **Ou seja, a principal parte desse trabalho se concentra nesses dois métodos**).

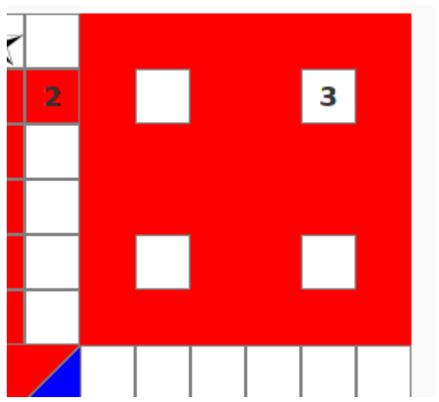
Quanto a classe *Casa*, existem vários tipos diferentes dela em todo o tabuleiro:



A tabela a seguir relaciona os retornos de vários métodos da classe com o número atribuído na imagem acima:

*	casaFinal()	pertenceGuarita()	ehEntradaDaZonaSegura()	getCor()	getCasaAnterior() == null	getCasaSeguinte() == null	getCasaSegura() == null
I	false	true	false	VERMELHO	true	true	true
II	false	false	false	VERMELHO	true	false	true
III	false	false	false	BRANCO	true	false	true
IV	false	false	true	BRANCO	true	false	false
V	false	false	false	AZUL	true	false	true
VI	false	false	false	AZUL	false	false	true
VII	true	false	false	AZUL	false	true	true

Finalmente, para uma melhor e correta interação com a GUI, recomenda-se o uso do método Casa.setQuantidadePecas(int quantidade) na casa final. Ao atribuir esta propriedade a uma casa (não necessariamente a casa final), a interface gráfica exibirá seu respectivo valor, desde que seja um inteiro maior ou igual a 2:



Por fim, o método Jogo.getJogadorDaVez() pode ser alterado para retornar a cor do jogador que deve agir no turno. Ao fazer isso, a GUI destacará a guarita desse jogador.