Uma Abordagem Baseada em Features Para o Desenvolvimento de Jogos de Tabuleiro Multiplataforma

Filipe Macêdo Borges Boaventura, *Autor*, Victor Travassos Sarinho, *Orientador*

***Resumo*—Diversas abordagens têm sido propostas para gerenciar a variabilidade do domínio dos jogos digitais em diferentes instâncias e estratégias. No entanto, a ideia de uma arquitetura *one-size-fits-all* para jogos digitais pode ser problemática, sendo preferível construir arquiteturas de jogos específicas para (sub)domínios específicos. Nesse sentido, este trabalho propõe uma abordagem baseada em features para desenvolver jogos de tabuleiro multiplataforma. Uma arquitetura de subdomínio de jogos, baseada em features identificadas de jogos de tabuleiro, é apresentada e implementada por artefatos adaptados para serem executados em plataformas de software distintas. Como resultado, será oferecida uma abordagem reutilizável para desenvolver jogos de tabuleiro multiplataforma, juntamente com o desenvolvimento de um jogo de tabuleiro para fins de validação.**

***Palavras-chave*—Engenharia de Software, Componentes de Software, Feature Model, Jogos Digitais**

***Abstract*—Several approaches have been proposed to manage the game domain variability in different instances and strategies. However, the idea of an one-size-fits-all game architecture can be misleading, being necessary to built reference game architectures for target (sub)domains. In that sense, this work proposes a feature-based approach to develop multiplatform board games. A subdomain game architecture, based on identified features of board games, is presented and implemented by feature artifacts adapted to be executed in distinct software platforms. As a result, a reusable approach to develop multiplatform board games will be provided, together with the development a board game for validation purposes.**

***Keywords*—Software Engineering, Software Components, Feature Model, Digital Games**

I. Introdução

O domínio dos jogos digitais representa "O domínio quintessencial para pesquisa e desenvolvimento da Ciência da Computação e Engenharia de Software"[1]. Isto é o resultado da atual demanda por habilidades técnicas e de integração em praticamente todas as principais áreas de pesquisa e educação em Ciência da Computação durante a produção de um jogo [1]. Como consequência, muitos dos grandes desafios tradicionais da Engenharia de Software surgem durante o desenvolvimento de jogos digitais como sistemas de software complexos, tais como engenharia de software em grande escala, engenharia de requisitos de jogos, design de software para jogos e assim por diante.

Em relação à engenharia de requisitos, o desenvolvimento de jogos digitais foca no levantamento e atendimento de *requisitos não-funcionais* (RNF), geralmente relacionados à aceitação do jogo por um publico-alvo em uma plataforma-alvo à uma faixa de preço ou esquema de monetização [2]. Com isso, o desafio da engenharia de software surge em determinar o que fazer durante o desenvolvimento para satisfazer a tais RNF como tarefas de engenharia [1].

Uma prática comum na indústria dos jogos digitais consiste na produção de jogos capazes de serem desenvolvidos incrementalmente e serem lançados com um conjunto mínimo de *features* que pode crescer adaptativamente para atender à RNFs informais [1]. Entretanto, como o desenvolvimento de jogos não é o mesmo que o desenvolvimento de software, a engenharia de requisitos tradicional não é aplicável [3] para determinar um conjunto adequado de *features* em um jogo. Portanto, é necessário seguir uma abordagem de desenvolvimento de jogos digitais capaz de sugerir *features* de jogo relacionadas à requisitos funcionais ou não funcionais.

Considerando os ambientes de desenvolvimento de jogos digitais, a reutilização de componentes específicos, como aqueles fornecidos por *SDKs*, *Frameworks* e *Game Engines* (motores de jogo), pode reduzir o tempo e o custo de desenvolvimento [4]. Entretanto, a seleção de um ambiente de desenvolvimento de jogos restringe ou predetermina que tipo de jogo digital pode ser desenvolvido [1]. Além disso, ambientes de desenvolvimento como motores de jogo geram uma alta dependência entre o jogo e os recursos do ambiente de desenvolvimento escolhido [5], algo que pode ser evitado com a separação do *core* dos objetos do jogo (*G-Factor*) da implementação propriamente dita, a fim de suportar a portabilidade do jogo entre os ambientes de desenvolvimento de jogos [5].

Diversas abordagens têm sido propostas para o desenvolvimento de jogos digitais, incluindo, por exemplo: linguagens de design e modelagem de jogos; meta-modelos; e *frameworks* [6]. Entretanto, a ideia de uma arquitetura universal, *"one-size-fits-all"*, para jogos digitais pode ser problemática, sendo preferível construir arquiteturas de jogos específicas para (sub)domínios específicos [3]. Com isso, este trabalho propõe o desenvolvimento de *features* para a produção de Jogos Digitais para múltiplas plataformas, porém voltados à um gênero específico de jogo: *Board Games* (Jogos de Tabuleiro). Como resultado, busca-se obter uma abordagem de desenvolvimento mais simples e escalável para Jogos de Tabuleiro em geral, através da padronização de *features* de Jogos de Tabuleiro em conjunto com os componentes de software necessários para a execução das configurações dos mesmos. A validação destes componentes se dará com a conclusão de versões digitais de Jogos de Tabuleiro similares aos encontrados no mercado,

consolidando, assim, uma Linha de Produto de Software [7] para a produção de Jogos de Tabuleiro.

O texto deste trabalho está organizado da seguinte forma. Na seção II, são abordados os referenciais teóricos necessários para produção deste trabalho, bem como alguns trabalhos relacionados. A seção III descreve a metodologia utilizada para a modelagem. A seção IV apresenta os resultados gerados neste trabalho, como o modelo de *features* proposto e a configuração do mesmo. Por fim, a seção V resume os principais pontos do trabalho e apresenta algumas possibilidades de trabalhos futuros.

II. Fundamentação Teórica

Esta seção apresenta a fundamentação teórica relevante ao trabalho, abordando, entre outros assuntos: desenvolvimento de jogos digitais, jogos de tabuleiro, desenvolvimento de software baseado em features e o uso dessa abordagem na produção de jogos digitais.

A. Desenvolvimento de Jogos Digitais

Segundo Costikyan [8], Jogos Digitais são softwares especializados que simulam disputas diversas e podem ser jogados em vários tipos de dispositivos eletrônicos (consoles, computadores pessoais, dispositivos móveis, etc).

Em relação às estratégias de desenvolvimento de jogos digitais, o lançamento do jogo *Quake* marcou o advento da abordagem de desenvolvimento baseada em *game engines* (motores de jogos), levando à uma mudança no paradigma de desenvolvimento de jogos digitais que deixaram de ser desenvolvidos do zero para utilizar dos componentes fornecidos por um motor de jogo escolhido [5], [9]. Com isso, atualmente é possível licenciar motores de jogos e reutilizar partes significativas de seus principais componentes de software para produzir jogos digitais [10]. Um Motor de Jogo pode ser definido como um software extensível que pode ser usado como base para muitos jogos diferentes sem grandes modificações [10]. Ainda sobre motores de jogos, estes representam a coleção de módulos de código de simulação que não especificam diretamente o comportamento do jogo (lógica do jogo) ou o ambiente do jogo (dados do nível) [9].

B. Desenvolvimento de Software Baseado em Features

Segundo Apel e Kästner [11], *Feature-Oriented Software Development* (FOSD) é um paradigma para a construção, customização e síntese de sistemas de software de larga escala em termos de *features*. O FOSD não é um método ou técnica de desenvolvimento único, mas um conglomerado de diferentes ideias, métodos, ferramentas, linguagens, formalismos e teorias, conectados pelo conceito de *feature* [11].

Kang et al. [12] foram os primeiros a introduzir o conceito de feature, apresentado originalmente como parte da *Feature-Oriented Domain Analysis* (FODA). Segundo Kang et al. [12], a *Modelagem de Features* é utilizada para identificar propriedades do sistema durante a análise de domínio. Um modelo de features representa as features padrões de uma família de sistemas e as relações entre elas [12]. Ainda segundo Kang et al. [12], *features* são aspectos ou características de um domínio que são visíveis ao usuário. Elas são usadas para identificar semelhanças ou diferenças entre os produtos de uma linha de produtos [13]. Por fim, Apel e Kästner [11] definem *feature* como uma unidade de funcionalidade de um sistema de software que satisfaz um requisito, representa uma decisão de projeto e fornece uma possível opção de configuração para ser composta como um sistema de software.

C. FOSD no Desenvolvimento de Jogos Digitais

Quanto ao uso de abordagens baseadas em features no desenvolvimento de jogos digitais, Zhang e Jarzabek [14] propuseram o *RPG Product Line Architecture* (RPG-PLA): um grupo de features comuns e variáveis de quatro jogos de RPG distintos. Como resultado, qualquer um dos RPGs originais, bem como outros semelhantes poderiam ser derivados do RPG-PLA.

Buscando identificar características comuns e reusáveis no domínio dos jogos digitais, Sarinho e Apolinário [15] propuseram o modelo de features *Narrative, Entertainment, Simulation e Interaction* (NESI). Trata-se de um modelo baseado na literatura referente a conceitos de jogos digitais capaz de representar o *G-factor* [5] no projeto de jogos digitais diversos.

No intuito de mostrar a viabilidade do uso de features na produção real de jogos digitais, Sarinho e Apolinário [16] também propuseram o modelo *GameSystem, DecisionSupport e SceneView* (GDS). Neste modelo, cada feature principal descreve configurações genéricas e aspectos comportamentais de um jogo, sendo estas focadas em aspectos de implementação identificados na literatura de jogos digitais [16]. A partir do modelo GDS, definiu-se também uma abordagem de produção generativa de jogos, tendo como base configurações de features do modelo GDS proposto.

Embora os modelos NESI e GDS sejam capazes de representar jogos digitas sem depender da estrutura de motores de jogos, o excesso de features propostas dificulta o desenvolvimento de jogos mais simples como os jogos casuais por exemplo. Uma prova desta dificuldade foi obtida com a produção do *Feature-based Environment for Digital Games* (FEnDiGa) [17], um motor baseado na integração de features NESI e GDS definidas via *Object Oriented Feature Modeling* (OOFM) [18]–[20]. Com isso, o *Minimal Engine for Digital Games* (MEnDiGa) foi proposto como uma coleção extensível de classes representativas, baseadas em um conjunto simplificado de features NESI e GDS, que podem ser usadas como base para jogos casuais e de pequeno porte sem grandes modificações [21].

D. Jogos de Tabuleiro

Os jogos de tabuleiro são um grupo específico de jogos em que figuras são manipuladas em um modo de jogo competitivo sobre uma superfície e de acordo com regras predefinidas [22]. Uma definição um pouco mais ampla é encontrada em [23], onde a expressão "jogo de tabuleiro" é usada para referir-se a jogos que se jogam sobre uma mesa, mesmo aqueles que não tenham tabuleiros, como os jogos de cartas clássicos, por

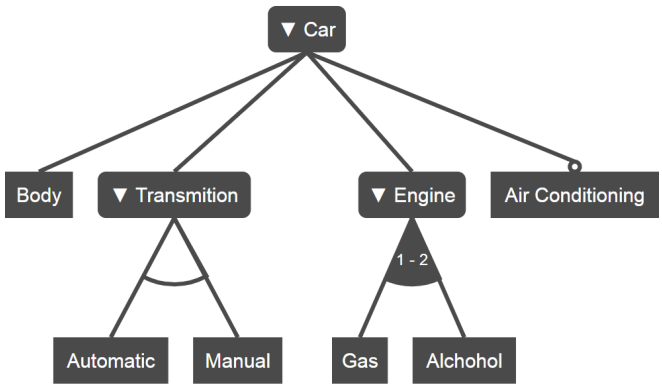

Figura 1. Exemplo de um *Feature Model* representando um carro.

exemplo. Jogos de tabuleiro geralmente contam com um plano delimitado dividido em setores onde um conjunto de peças, associadas aos jogadores por forma ou cor, são movimentadas de acordo com os eventos do jogo [24].

Os objetivos dos jogo de tabuleiro, bem como suas estratégias e condições de vitória, podem ser bastante diversos como, por exemplo, conquistar a maior quantidade de áreas do tabuleiro, eliminar as peças dos jogadores adversários, acumular mais pontos ou outra forma de moeda [24]. Quanto às diversas mecânicas presentes nos jogos de tabuleiro, tem-se no *Board Game Geek* (BGG) um dos maiores e mais usados fóruns sobre Jogos de Tabuleiro na internet [25]. Este site se dedica a catalogar informações sobre jogos de tabuleiro físicos, no intuito de manter uma base de dados para fins de posteridade e pesquisa histórica [26]. Contendo uma listagem de 99953 Jogos de Tabuleiro, 83 categorias e 51 mecânicas de jogo [26], a base de dados do BGG se tornou uma fonte de informações muito utilizada tanto por *game designers* quanto por pesquisadores [25].

Com base nas informações disponíveis na base de dados do BGG, Kritz, Mangeli, e Xexéo [25] propuseram uma ontologia de mecânicas de jogos de tabuleiro, seguindo o conceito de Mecânica (Mechanics) apresentado no *MDA (Mechanics, Dynamics e Aesthetics) Framework* [27]. O MDA é uma abordagem formal para analisar jogos, construída com o objetivo de compreender os conceitos que ajudam designers, pesquisadores e estudiosos a realizar a decomposição de jogos em partes coerentes e compreensíveis [27]. Segundo Hunicke, Leblanc e Zubek [27], Mecânicas descrevem componentes particulares de jogos no nível de representação de dados e algoritmos.

Com isso, a ontologia proposta em [25] organiza diversas mecânicas de jogos de tabuleiro [26] em dois principais conceitos derivados de [27]: *Algorithm mechanics* e *Data Representation mechanics*. *Algorithm* é a mecânica geral para os processos que acontecem no jogo e está subdividida em: *Action* (Ações) — conjunto de mecânicas que permitem que o usuário interaja com o jogo; *Ruleset* (Regras) — mecânicas que definem, entre outros aspectos, o comportamento dos componentes do jogo; e *Goal* (Objetivos) — mecanicas que definem os diversos objetivos a serem alcançados durante a partida, como condições de vitória ou metas transitórias [25]. Já o *Data Representation* é a mecanica geral para armazenar e transmitir informações nos jogos e está dividia em: *Component* — representando os elementos do jogo que os jogadores podem possuir e manipular diretamente; e *Resource* — representando os elementos do jogo que o jogador deve gerenciar para cumprir objetivos (*Goals*) [25].

Por fim, o *Quiz Board Game Model* propôs um modelo formal para representar *quizzes* como jogos de tabuleiro com apelo multimídia [22]. Este modelo descreve jogos de tabuleiro como um conjunto contendo, entre outros elementos: objetos, posições, funções, ações, regras e efeitos [22]. O jogo é apresentado sobre uma imagem de fundo, onde os objetos de diversos tipos são dispostos nas posições (que também podem assumir diversos tipos) de acordo à uma matriz que descreve a disposição inicial dos mesmos [22]. As questões do quiz são dispostas nas posições do tabuleiro, sendo apresentadas ao jogador de uma maneira atraente, com multimídia avançada, a medida que os objetos encontram essas posições[22].

III. Metodologia

O trabalho desenvolvido neste artigo segue um paradigma *FOSD* [11] em que um *modelo de features* (*Feature Model* [12]) é usado para descrever "os relacionamentos e dependências de um conjunto de features pertencentes a um domínio específico" [11]. Os modelos foram desenhados usando a ferramenta *Glencoe* [28], sendo que esta utiliza uma notação

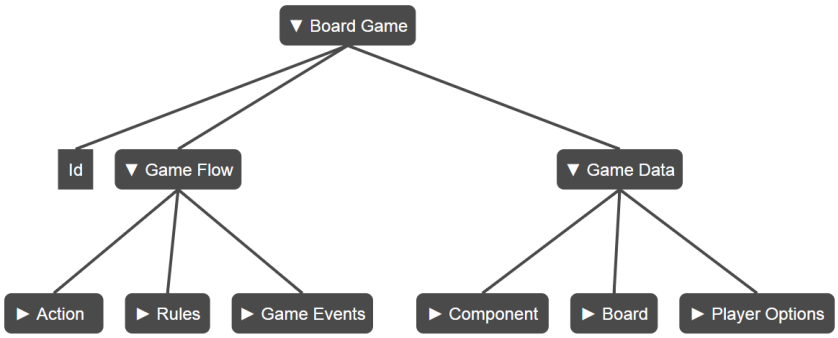

Figura 2. Diagrama de Features Parcial Para Jogos de Tabuleiro.

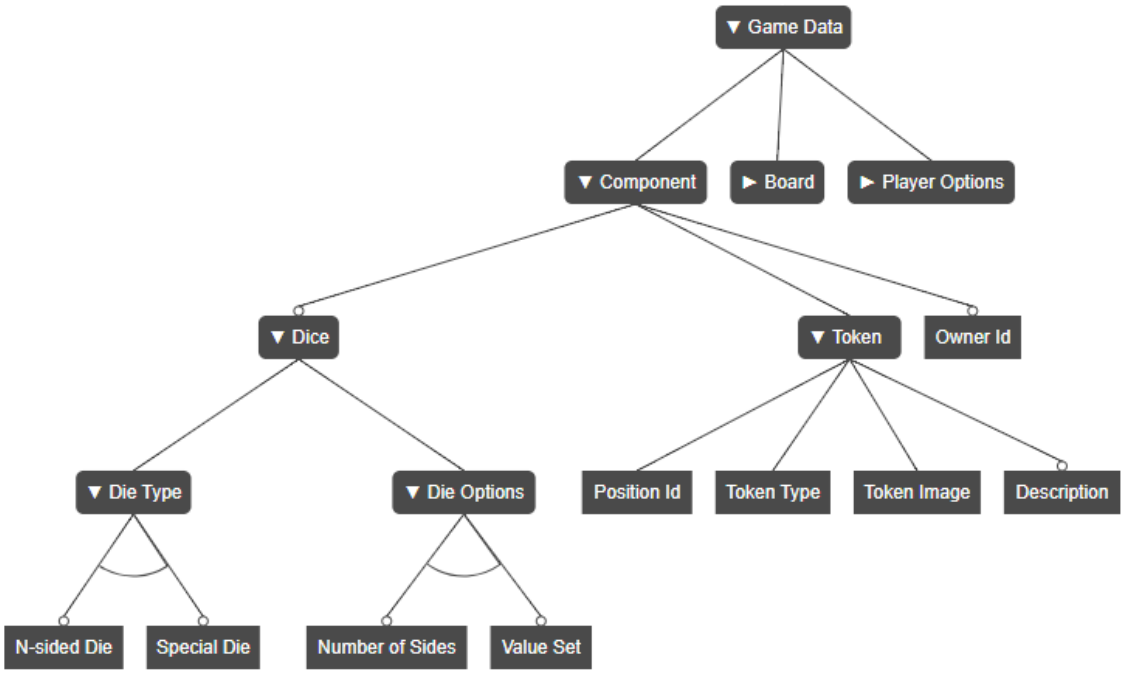

Figura 3. *Game Data subfeatures*, focando em *Component*.

baseada na notaçao *FODA*, introduzida por Kang et al [12]. Na notação utilizada pelo *Glencoe*, um modelo de features é representado como uma hierarquia de featrues, organizadas em uma estrutura em árvore. A raiz da arvore representa o conceito que está sendo modelado, enquanto que os demais nós representam features. *Subfeatures* são conectados aos seus *parent features* por linhas sólidas. Um circulo vazio ao fim de uma linha indica uma *feature opcional*. Subfeatures também podem estar agrupadas em grupos *OR* (arco vazio) ou *XOR* (arco preenchido). Apenas uma feature de um grupo OR pode fazer parte de uma configuração específica do sistema em um dado momento, ao contrário do grupo XOR, que permite que uma ou mais features sejam utilizadas ao mesmo tempo. Na Figura 1 é mostrado como exemplo um modelo de features simples, descrevendo as possíveis configurações de um carro.

IV. Resultados

A. *Features para Jogos de Tabuleiro*

Com os conhecimentos adquiridos a partir dos estudos realizados, foi possível dar início à modelagem dos componentes

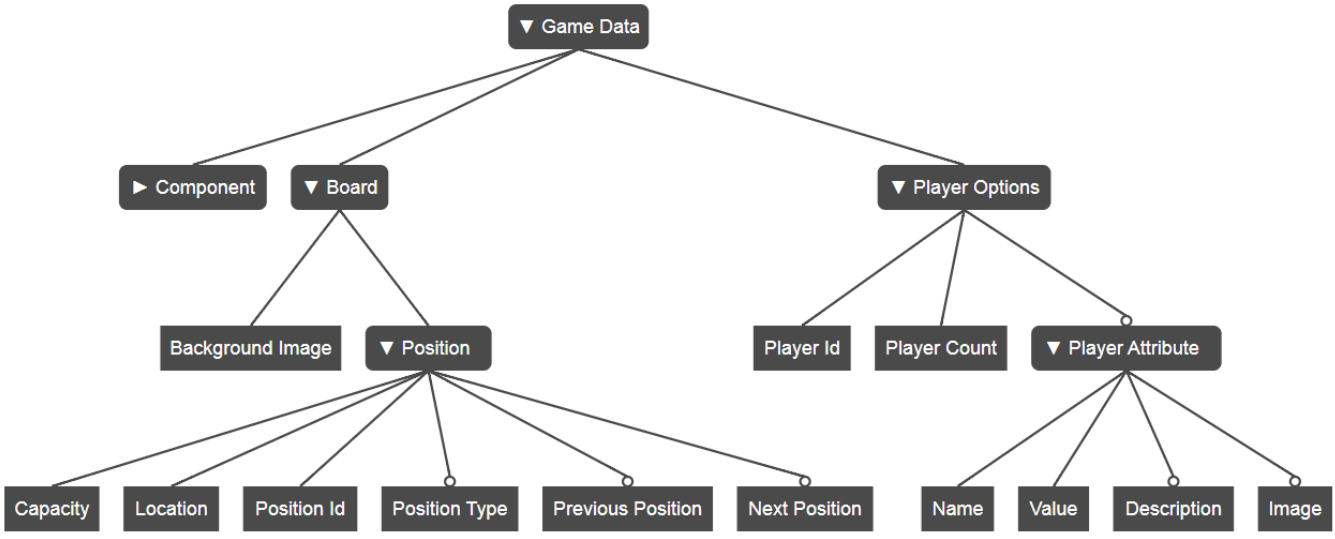

Figura 4. *Game Data subfeatures*, focando em *Board* e *Player Options*.

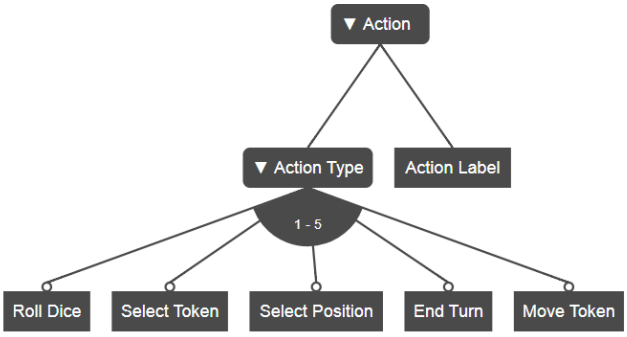

Figura 5. *Action* subfeatures.

de software genéricos para a produção de jogos de tabuleiro digitais. Estes componentes foram projetados de acordo com os conceitos de Jogos de Tabuleiro encontrados na literatura [22], [25], [26], e tem, como objetivo, representar jogos digitais de forma independente de seus recursos de implementação, garantindo assim a portabilidade e representatividade padronizada do negócio do jogo (*G-Factor*) [5].

Um modelo contendo atualmente 79 features foi proposto para representar os componentes de jogos de tabuleiro propostos. O modelo proposto apresenta features responsáveis por descrever os componentes que definem o fluxo do jogo (*Game Flow*) e a representação de informações manipuladas durante a partida (*Game Data*). *Game Flow* é representado em termos de *ações* (*Action*), *regras* (*Rules*) e *eventos* (*Game Events*), enquanto que *Game Data* descreve os *componentes* (*Component*), o *tabuleiro* (*Board*) e as *opções de jogador* (*Player Option*) do jogo. Um diagrama com as principais features é exibido na Figura 2.

Quanto às subfeatures de *Game Data* (Figura 3), *Component* representa os objetos que podem ser manipulados pelos jogadores durante as partidas [25]. Estes objetos podem ser *dados* (*Dice*) ou *Tokens* de diversos tipos (*Token Type*). *Tokens* possuem componentes para representação visual (*Token Image*) e textual (*Description*) e são posicionados no tabuleiro de acordo com seu *Position Id*. *Dice* aparecem em dois tipos distintos: *N-sided Die* representam dados comuns, contendo uma sequencia de números de 1 à N; enquanto que o *Special Die* pode ser usado para representar um dado personalizado pelo *game designer*, contendo uma lista de valores personalizados (*Value Set*). *Components* podem pertencer à um jogador específico, identificado pela feature *Owner Id*.

Ainda sobre *Game Data* (Figura 4), temos na feature *Board* a representação do tabuleiro do jogo. O tabuleiro é composto por uma *imagem de fundo* (*Background Image*) e um conjunto de *posições* (*Position*) que, assim como *Dice*, podem assumir diversos tipos (*Position Type*). *Positions* são identificadas por um *Position Id* e são localizadas por um *Location* feature, representando as coordenadas cartesianas da posição no tabuleiro. Além disso, posições também podem fazer referência à outras posições adjacentes (*Previous Position* e *Next Position*), formando a trilha que deve ser percorrida pelos *Tokens*. Tanto *Previous Position* quanto *Next Position* podem fazer referência a mais de uma posição, permitindo a formação de múltiplos caminhos no tabuleiro. *Positions* acomodam um ceto numero de *Tokens* de acordo com sua *capacidade* (*Capacity*). Por fim, *Player Options* representam as configurações do jogo quanto ao número de jogadores (*Player Count*), seus identificadores (*Player Id*) e seus atributos (*Player Attribute*). *Player Attribute* representa uma lista contendo os diversos tipos de recursos que os jogadores podem acumular durante o jogo, como, por exemplo, pontuação e valores utilizados como moeda. Estes recursos são representados por nome (*Name*), valor (*Value*), uma descrição textual (*Description*) e uma imagem ilustrativa (*Image*).

Quanto às subfeatures de *Game Flow*, temos em *Action* (Figura 5) o conjunto de ações que podem ser realizadas durante o jogo. Cada tipo de ação (*Action Type*) é apresentada para o jogador por um *Action Label*, que contém uma descrição textual da ação. É através dessas ações que o jogador é capaz de alterar o estado do jogo, manipulando *Components* e/ou *Player Attributes*. Por exemplo, através da feature *Roll Dice*, o jogador é capaz de interagir com o *Component Dice* para gerar números aleatórios. *Select Token* e *Select Position* permitem que o jogador marque um *Token* ou uma posição respectivamente. *End Turn* permite que o jogador encerre sua jogada, dando a vez para o próximo jogador. Por fim, a feature *Move Token* é utilizada para mover *Tokens* no tabuleiro, de acordo com as regras que serão discutidas mais a frente.

A execução de ações e a checagem de regras podem disparar diversos eventos do jogo (*Game Events*, figura 6). Atualmente estes eventos incluem: *Dice Event* — acionado quando um valor é recuperado de um dado; *Passing Event* e *Stopping Event* — acionados quando um *Token*, respectivamente, passa ou para uma posição; *End Turn* e *End Game* — acionados

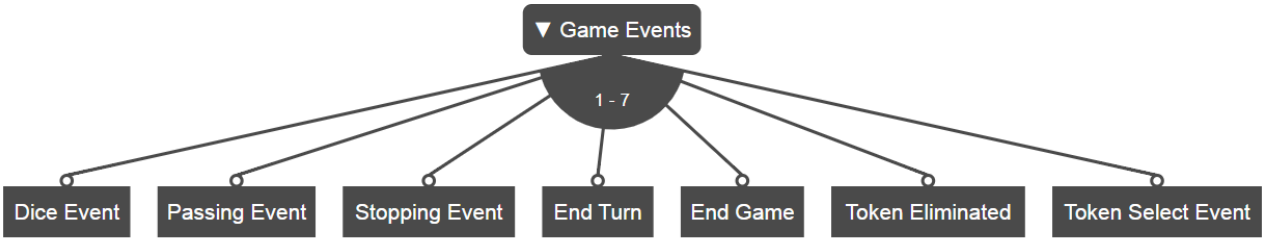

Figura 6. *Game Events* subfeatures.

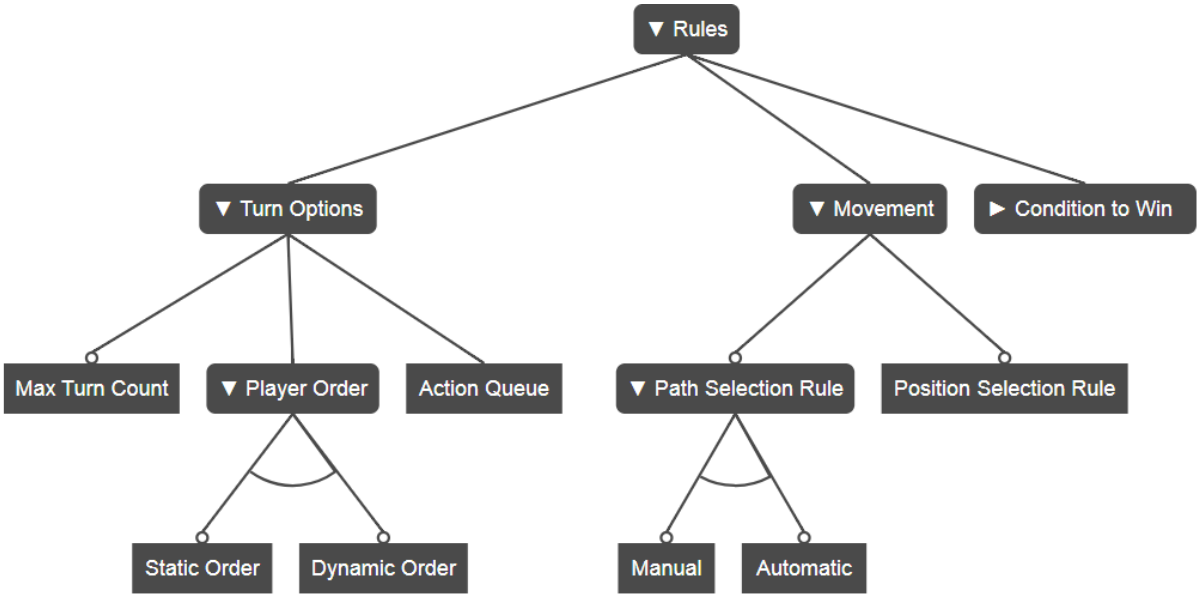

Figura 7. *Rule* subfeatures, focando em *Turn Options* e *Movement*.

respectivamente quando um turno ou o jogo é concluído; *Token Eliminated* — acionado quando o jogador perde um *Token*; e *Token Select Event* — acionado quando o jogador seleciona um *Token*.

Rule features (figura 7) representaram as regras de jogo, sendo responsáveis por avaliar a possibilidade de execução de *Action* features de acordo com o estado atual do jogo, bem como definir a configuração de outros aspectos do jogo, como a movimentação de *Tokens* (*Movement*), as políticas de sucessão de jogadores ao longo dos turnos (*Turn Options*) e as condições de vitória (*Condition to Win*) do jogo.

Quanto às regras relacionadas aos turnos, temos em *Turn Options* as configurações do jogo quanto ao numero máximo de turnos (*Max Turn Count*), a ordem dos jogadores (*Player Order*) e a ordem das ações a serem realizadas durante um turno (*Action Queue*). A sucessão dos jogadores pode ser feita

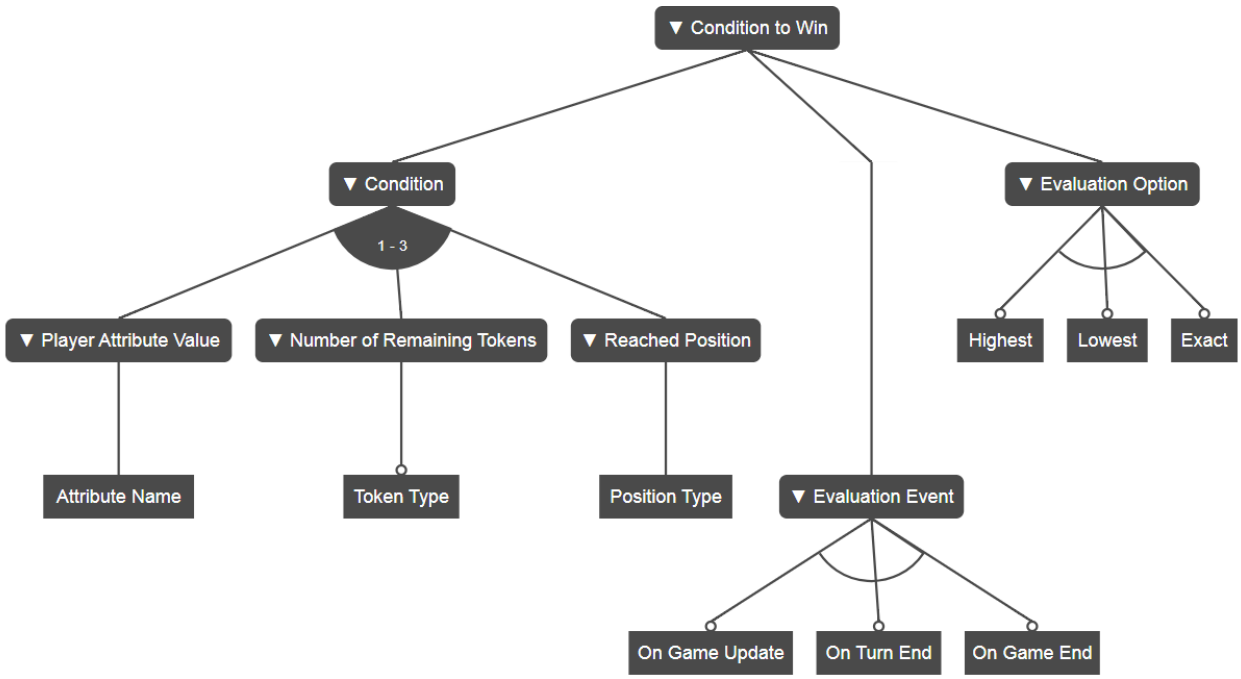

Figura 8. *Condition to Win* subfeatures.

```
gameFlow: {
  actions: [
    {actionType: "rollDice", actionLabel: "Roll Dice"},
    {actionType: "selectToken", actionLabel: "Select Token to move"},
    {actionType: "moveToken", actionLabel: ""}
  ],
  rules: {
    movement: {
      pathSelectionRule: function (GameStatus) {...} // Ou "manual" ou uma funcao
    },
    turnOptions: {
      maxTurnCount: null,
      playerOrder: function (GameStatus){...},
      actionQueue:["rollDice", "selectToken", "moveToken"]
    },
    conditionToWin: {
      numRemainingTokens: {tokenType:null, evalOption:"exact", value:0, evalEvent:"update"}
    }
  },
  gameEvents: {
    tokenSelected: function(GameStatus, selectedToken) {
      if (
        selectedToken.position.positionType.includes("Base") &&
        GameStatus.currentPlayer.diceValue != 6 &&
        GameStatus.currentPlayer.diceValue != 1 &&
        GameStatus.currentPlayer.attributes["Active Tokens"] > 0
      ) {
        GameStatus.repeatAction(
          "You got a " + GameStatus.currentPlayer.diceValue}.
          + " You need a 1 or a 6 to move a token into the game."
          + " Please select another token.");
      }
    }, ...
  }
```

Figura 9. JSON representando uma configuração parcial de um jogo de tabuleiro.

em uma ordem fixa (*Static Order*) ou pode variar de acordo com algum evento (*Dynamic Order*). Por exemplo, um jogo pode dar a vez para um jogador que acabou de finalizar seu turno se este cumpriu alguma condição especial.

Quanto à movimentação (*Movement*), esta é diretamente afetada pela organização da posições no tabuleiro. Em jogos onde as posições são organizadas como uma trilha de pontos interligados, os *Tokens* podem apenas se mover para a próxima posição adjacente à sua posição atual (*Next Position*, figura 3). Este tipo de movimento é conhecido como *Point to Point Movement* [25], [26]. Caso a posição atual do *Token* faça referência para mais de uma posição adjacente, é necessário decidir para qual posição o *Token* deve ser movido. Esta decisão é configurada pela feature *Path Selection Rule*, que pode ser: *Manual*, delegando ao jogador a tarefa de selecionar a posição desejada; ou *automática*, onde o jogo decide para onde o *Token* deve ser mandado. Quanto ao *Position Selection Rule*, este representa um verificador a ser fornecido pelo *game designer* para validar uma posição selecionada pelo jogador.

Condition to Win (figura 8) representa uma coleção com as condições de vitória do jogo, bem como as configurações de avaliação das mesmas. Das condições disponíveis atualmente, *Reached Position* se refere ao ato de mover um *Token* para uma posição com um *Position Type* específico. *Player Attribute Value* e *Number of Remaining Tokens* são condições relacionadas a valores numéricos e se referem, respectivamente: a um *Player Attribute*, identificado por um *Attribute Name*; ou a quantia *tokens* restantes de um certo tipo (*Token Type*). Estas condições podem ser avaliadas quanto ao maior valor (*Highest*), menor valor (*Lowest*) ou comparadas com um valor exato (*Exact*), como descrito pela feature *Evaluation Option*. Por fim, a feature *Evaluation Event* define o momento em que a checagem das condições é realizada. As condições podem ser avaliadas para toda atualização do estado do jogo (*On Game Update*) ou apenas uma vez a cada turno (*On Turn End*). Em ambos os casos, o jogo termina se o vencedor for encontrado. Caso o jogo termine sem um vencedor, por exemplo, se número máximo de turnos for atingido, as condições de vitória de fim de jogo (*On Game End*) são avaliadas para definir o vencedor.

B. Desenvolvimento de Jogos usando Board Game Features

Com a definição das features propostas em mãos, é necessário definir uma abordagem de desenvolvimento para a implementação de jogos de tabuleiro desejados. Diversas abordagens de desenvolvimento de software baseado em features (*FOSD* [11]) foram utilizadas com sucesso para o desenvolvimento de jogos digitais [3], [16]. Para este trabalho, uma *Domain-Specific Language* (*DSL*) foi definida para representar as features propostas. Além disso, um *game loop* genérico

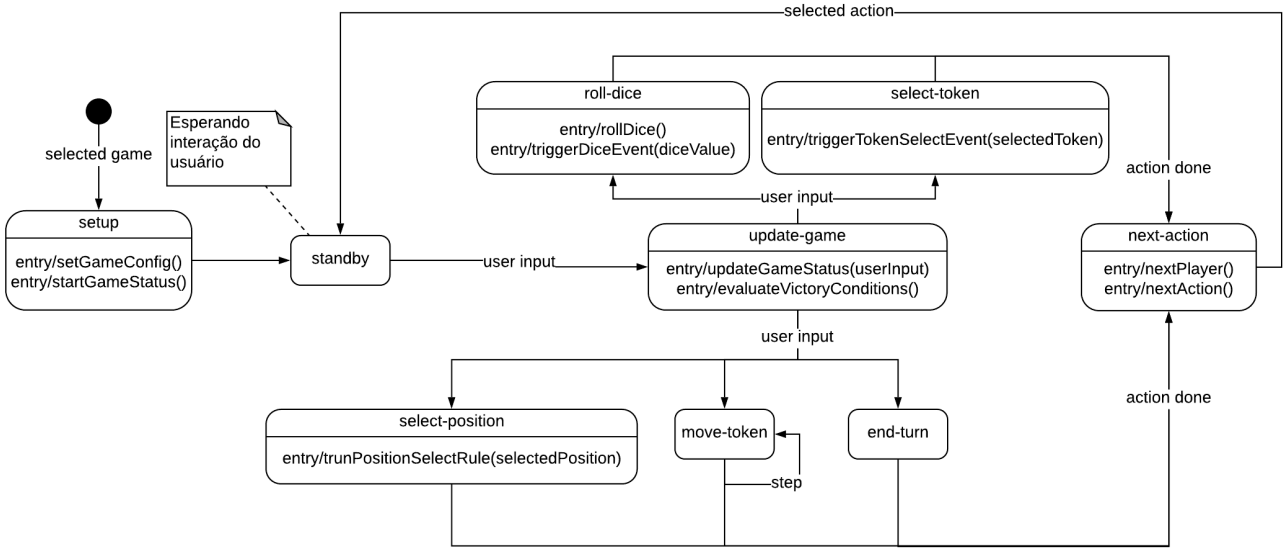

Figura 10. Diagrama de estados do *game loop* proposto.

foi desenvolvido para interpretar e executar configurações de jogos de tabuleiro.

Uma *DSL* é uma linguagem específica para um domínio em questão em que abstrações pré-definidas são fornecidas para representar os conceitos do domínio [29].

Através do modelo de features proposto, uma DSL foi definida usando *JavaScript Object Notation* (*JSON*), onde cada feature foi usada para representar os valores numéricos, textuais e procedurais envolvidos na configuração de um jogo de tabuleiro digital. A Figura 9 ilustra um exemplo de configuração parcial da feature *Game Flow* e suas subfeatures *Actions*, *Rules* e *Game Events*.

O *game loop* é um algoritmo que relaciona o estado atual do jogo e as propriedades de seus objetos com diversas condições capazes de modificar o estado do jogo [30]. Ele é executado repetidamente, atualizando diversos sistemas do jogo, como inteligência artificial e simulações físicas, a cada iteração [10].

Para fornecer um *game loop* capaz de executar as features propostas, foi implementado um interpretador em *Javascript* capaz de executar as ações, eventos e controlar o fluxo do jogo definido no documento JSON. Chamado de *BoardGameAdapter*, o interpretador fornece uma interface adaptação que permite a inicialização e a atualização o jogo, usando as funções *startGameStatus* e *updateGameStatus* respectivamente. A função *updateGameStatus* executa as ações do jogo na ordem definida na *Action Queue* (figura 9) além de acionar os *Game Events* apropriados quando necessário. Cada ação executada é removida da fila. Quando não há mais ações a serem executadas, o turno é encerrado, a fila de ações é restaurada e o jogo identifica o próximo jogador. O diagrama de estados do *BoardGameAdapter* é mostrado na figura 10.

C. Clientes Multiplataforma para Jogos de Tabuleiro

As ferramentas de desenvolvimento de software multiplataforma estão ficando cada vez mais populares devido à sua característica de compilar o código-fonte do aplicativo para diversas plataformas suportadas, especialmente para sistemas operacionais *mobile* [31], [32]. Componentes de software desenvolvidos em *FOSD* encapsulam informações de design e implementação, permitindo uma geração quase automática de aplicativos devido ao mapeamento nítido com as features representativas dos conhecimentos do domínio [11]. Com isso, as *Board Game Features* propostas foram implementadas levando em consideração apenas os conceitos relacionados ao domínio dos jogos de tabuleiro, sem uma plataforma específica em mente. Elas seguem a estrutura do documento JSON proposto (*Model*) e junto com o *BoardGameAdapter* (*Controller*), permitem que clientes (*View*) sejam implementados de forma independente para plataformas específicas.

Para este trabalho, foi implementado um cliente web (figura 11) usando o *Phaser*, um *framework open source* para jogos de navegador baseados em *Canvas* e *WebGL*. A interface do usuário é dividida em três áreas principais: o *sidebar* esquerdo exibe os *Player Attributes* do jogador atual, bem como seu ícone ilustrativo e descrição textual, sendo este ultimo exibido no evento de *mouse over*; a área central exibe, entre outras informações, o tabuleiro do jogo, desenhado pelo *Phaser* em um *Canvas* com resolução de 800x600; e o *sidebar* direito exibe as ações disponíveis e mensagens de *feedback* para o jogador. As entradas do usuário são obtidas através de eventos de clique em *hyperlinks* no *sidebar* direito ou nos tokens do *Canvas* central. Estes eventos executam o *updateGameStatus* do *BoardGameAdapter* que por sua vez, executa a ação apropriada de acordo com o estado atual do jogo. Após a execução do *updateGameStatus*, o *BoardGameAdapter* notifica o cliente

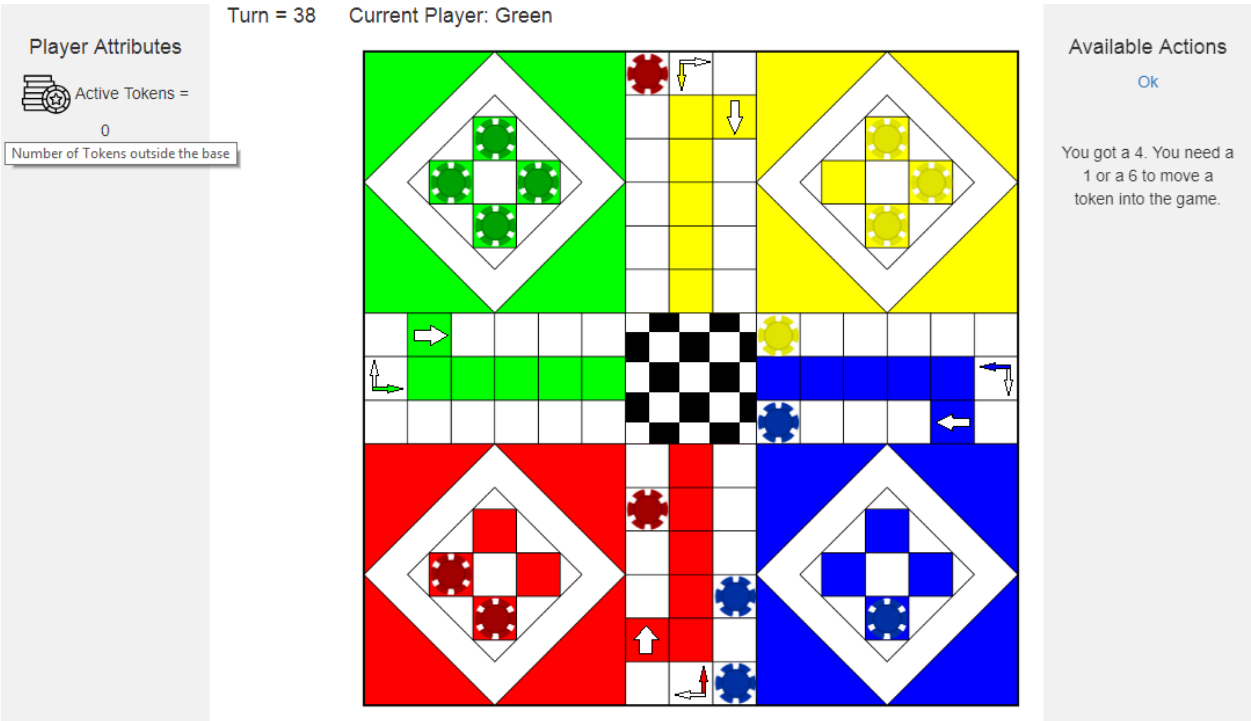

Figura 11. O jogo Ludo sendo executado no cliente Web.

para atualizar a *View* com os novos valores do estado atual do jogo.

D. Validação

Para a validação das features de Jogos de Tabuleiro propostas, foi implementado uma versão digital do jogo *Ludo*: um jogo de tabuleiro para quatro pessoas, baseado no jogo *Pachisi*, de origem indiana [33]. Cada jogador é representado por uma cor e possui quatro tokens, localizados inicialmente em posições específicas que chamaremos de *bases*. O objetivo do jogo é levar os tokens da base para a linha de chegada no centro do tabuleiro. O movimento é controlado por um dado simples de seis lados. O jogador lança o dado e escolhe um token fora da base para mover tantas posições quanto o valor obtido pelo dado. Entretanto, para mover um token da base é preciso tirar um ou seis no dado. Além disso, o jogador que tira seis tem o direito de lançar o dado mais uma vez. Esta jogada pode ser repetida por duas vezes e se o jogador tirar um seis pela terceira vez, ele passa a vez para o próximo jogador.

As posições do tabuleiro acomodam apenas um token por vez. Caso um token pare em uma posição que contenha um token adversário, este ultimo é ”capturado” e mandado de volta a sua base. Caso o token que ocupe a posição não seja do adversário, o movimento do token que chega é desfeito, mandando ele de volta para a posição que estava no início do turno.

Os tokens devem percorrer o tabuleiro no sentido horário até chegar em sua respectiva *zona segura*: uma parte do tabuleiro em que as posições tem a mesma cor do token. Tokens não podem entrar na zona segura de um adversário, tornando a captura impossível. Uma vez dentro da zona segura, o jogador tem que tirar no dado o numero exato de posições que faltam para mandar o token para a linha de chegada. Caso tire um número maior, o token avança para a posição final, volta para o início da zona segura e avança o número de posições que sobraram. Por fim, ganha o jogador que conseguir colocar todos os seus tokens na posição final.

A partir dessas informações, podemos configurar os componentes do jogo em um documento JSON, como descrito na seção IV-B e como pode ser visto nas figuras 9 e 12.

Na figura 9, entre outras informações, é possível ver a definição da fila de de ações (*actionQueue*), a definição de uma condição de vitória (*conditionsToWin*) e a implementação de um *gameEvent*. O *gameEvent* em questão é o *tokenSelect* e ele é responsável por garantir que tokens não possam sair da base caso o jogador tenha tirado um valor diferente de um ou seis. Quanto a condição de vitória, o jogo foi programado para remover do jogador o token que parou na posição final, usando o *stoppingEvent*. Portanto, a condição de vitória é alcançada pelo primeiro jogador que perde todos os seus tokens (*numRemainingTokens = 0*).

Na figura 12, é possível ver como são definidos os identificadores dos jogadores, bem como seus atributos. O atributo *combo* é utilizado como um contador para registrar quantas vezes o jogador tirou um seis em um dado. Este atributo é atualizado a cada *Dice Event* e seu valor é avaliado pela função *playerOrder* (figura 9), responsável por decidir o próximo jogador. A *flag visible* serve para informar ao cliente se o valor deste atributo deve ou não ser ser visível para o jogador. Quanto as posições do tabuleiro Ludo, na entrada

```
gameData: {
  playerOptions: {
    playerCount: 4,
    playerId: ["Red", "Green", "Yellow", "Blue"],
    playerAttribute: [
      {name: "Active Tokens", value: 0,
        description: "Number of Tokens outside the base",
        image: "assets/imgs/tokenpile.svg", visible:true},
      {name: "combo", value: 0, description: "", image: "", visible:false}
    ]
  },
  board: {
    background:"assets/imgs/LudoBoard2.png",
    positions: [
      {capacity:1, location:[0.171, 0.500], positionId:11,
        positionType:"green", prev: [10], next: [12,57]},
      ...
    ]},
  component: {
    dice: [{dieType:"nSidedDie", numberOfSides:6}],
    token: [
      {positionId: 73, tokenType: "red",
        tokenImage: "assets/imgs/pokerchip1.png", ownerId:"Red"},
      ...
    ]} ...
```

Figura 12. Configuração parcial da feature *Game Data* para o jogo Ludo.

da zona seugura, estas formam bifurcações conforme as setas desenhadas no tabuleiro. A função *pathSelectionRule* (figura 9) é responsável por organizar a passagem de tokens por estas posições, encaminhando tokens da cor apropriada para a zona segura.

Por fim, o arquivo de configuração JSON é usado pelo *BoardGameAdapter* para fornecer ao cliente web as informações necessárias para a exibição do jogo final. Uma captura de tela do jogo Ludo é mostrada na figura 11.

V. Conclusões

Neste trabalho, foi apresentada uma abordagem baseada em *features* para o desenvolvimento de jogos de tabuleiro digitais. Foi definido um modelo de features para representar diversos conceitos do domínio dos jogos de tabuleiro, juntamente com os artefatos de software capazes de configurar e executar as dinâmicas do jogo em clientes multiplataforma.

Esta abordagem tem como objetivo desacoplar a lógica de negócio do jogo dos recursos de implementação fornecidos por ferramentas focadas em plataformas específicas. Uma vez que os componentes de um jogo de tabuleiro são identificados e configurados através da DSL proposta, estes podem ser executados pelo *BoardGameAdapter* que, por sua vez, é executado em uma aplicação cliente desenvolvida para uma plataforma específica. Dessa forma, para desenvolver um novo jogo para um cliente existente, basta criar um novo arquivo de configuração JSON descrevendo-o. Além disso, é possível implementar novos clientes especializados para novas plataformas, sem que seja necessário modificar o *BoardGameAdapter* e os jogos configurados pela DSL.

Por fim, uma implementação do jogo *Ludo* também foi apresentada neste trabalho no intuito de mostrar a viabilidade dos artefatos produzidos no desenvolvimento de versões digitais de jogos de tabuleiro existentes.

Para trabalhos futuros, serão desenvolvidos mais jogos usando os artefatos propostos no intuito de realizar uma análise comparativa. Pretende-se coletar métricas de software [34] relacionadas ao reuso e complexidade de código para avaliar a reusabilidade e a *maintainability* de cada jogo desenvolvido com os artefatos de software propostos. Além disso, pretende-se expandir o modelo de features proposto para contemplar as mecânicas de jogos de tabuleiro que não etão presentes no modelo atual, como, por exemplo, as mecânicas de movimentação em tabuleiros do tipo *grid*. A implementação de partidas *multiplayer* também serão consideradas em trabalhos futuros.

VI. Agradecimentos

Primeiramente a Deus, por ter permitido que eu chegasse até aqui. Por ter me presenteado com uma família e amigos incríveis que estão sempre nos meus pensamentos e no meu coração.

Aos meus pais, pelo amor, apoio e carinho incondicional. Sem vocês eu jamais teria chegado até aqui.

Ao meu orientador, Victor Sarinho, por sua paciência e seus ensinamentos ao longo de mais de dois anos de iniciação científica e diversos trabalhos.

E a todos os meus amigos, que sofreram e sorriram comigo ao longo dessa jornada, o meu muito obrigado.

Referências

[1] W. Scacchi and K. M. Cooper, “Research challenges at the intersection of computer games and software engineering,” in *Conference on Foundations of Digital Games (FDG 2015)*, Pacific Grove, CA, June 2015.

[2] D. Callele, E. Neufeld, and K. Schneider, "Requirements engineering and the creative process in the video game industry," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, Aug 2005, pp. 240–250.

[3] A. W. Furtado, A. L. Santos, G. L. Ramalho, and E. S. de Almeida, "Improving digital game development with software product lines," *IEEE software*, vol. 28, no. 5, pp. 30–37, 2011.

[4] E. Folmer, "Component based game development: A solution to escalating costs and expanding deadlines?" in *Proceedings of the 10th International Conference on Component-based Software Engineering*, ser. CBSE'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 66–73.

[5] A. BinSubaih and S. Maddock, "Game portability using a service-oriented approach," *Int. J. Comput. Games Technol.*, vol. 2008, pp. 3:1–3:7, Jan. 2008.

[6] S. Tang and M. Hanneghan, "State-of-the-art model driven game development: A survey of technological solutions for game-based learning," *Journal of Interactive Learning Research*, vol. 22, no. 4, pp. 551–605, 2011.

[7] L. M. Northrop, "Software product lines: reuse that makes business sense," in *Australian Software Engineering Conference (ASWEC'06)*, April 2006, pp. 1 pp.–3.

[8] G. Costikyan, "I have no words & I must design," 1994, *Accessed 26 mai. 2010.* [Online]. Available: http://www.costik.com/nowords.html

[9] M. Lewis and J. Jacobson, "Games engines in scientific research," *Communications of the ACM*, vol. 45, no. 1, pp. 27–31, 2002.

[10] J. Gregory, *Game engine architecture*. AK Peters/CRC Press, 2014.

[11] S. Apel and C. Kästner, "An overview of feature-oriented software development." *Journal of Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

[12] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-90-TR-021, 1990.

[13] M. Antkiewicz and K. Czarnecki, "Featureplugin: Feature modeling plug-in for eclipse," in *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange*, ser. eclipse '04. New York, NY, USA: ACM, 2004, pp. 67–72.

[14] W. Zhang and S. Jarzabek, "Reuse without compromising performance: Industrial experience from rpg software product line for mobile devices," in *Proceedings of the 9th International Conference on Software Product Lines*, ser. SPLC'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 57–69.

[15] V. Sarinho and A. Apolinário, "A feature model proposal for computer games design," in *VII Brazilian Symposium on Computer games and Digital entertainment, Belo horizonte*, 2008, pp. 54–63.

[16] V. T. Sarinho and A. L. Apolinário, "A generative programming approach for game development," in *Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on*. IEEE, 2009, pp. 83–92.

[17] V. T. Sarinho, A. L. Apolinário, and E. S. Almeida, "A feature-based environment for digital games," in *Entertainment Computing - ICEC 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 518–523.

[18] V. T. Sarinho and A. L. Apolinário, "Combining feature modeling and object oriented concepts to manage the software variability," in *Information Reuse and Integration (IRI), 2010 IEEE International Conference on*. IEEE, 2010, pp. 344–349.

[19] V. T. Sarinho, A. L. Apolinário, and E. S. d. Almeida, "Oofm - a feature modeling approach to implement mpls and dspls," in *2012 IEEE 13th International Conference on Information Reuse Integration (IRI)*, Aug 2012, pp. 740–742.

[20] V. Sarinho and A. Apolinário, "Detailing the uml profile of the oofm technique," in *of the 3rd Brazilian Workshop on Model Driven Development (WB-DSDM'12)*, vol. 8, 2012, pp. 25–32.

[21] F. Boaventura and V. T. Sarinho, "Mendiga: A minimal engine for digital games," *International Journal of Computer Games Technology*, vol. 2017, 2017.

[22] B. Bontchev and D. Vassileva, "Educational quiz board games for adaptive e-learning," in *Proc. of Int. Conf. ICTE*, 2010, pp. 63–70.

[23] L. C. S. Duarte and S. Federal, "Jogos de tabuleiro no design de jogos digitais," in *Anais do XI Simpósio Brasileiro de Jogos e Entretenimento Digital, Brasília, DF*, 2012, pp. 132–137.

[24] F. Lucchese and B. Ribeiro, "Conceituação de jogos digitais," 2009, *Accessed 12 nov. 2018.* [Online]. Available: http://www.dca.fee.unicamp.br/~martino/disciplinas/ia369/trabalhos/t1g3.pdf

[25] J. Kritz, E. Mangeli, and G. Xexéo, "Building an ontology of board-game mechanics based on the boardgamegeek database and the mda framework," in *XVI Brazilian Symposium on Computer Games and Digital Entertainment, Curitiba*, 2017, pp. 182–191.

[26] BoardGameGeek, "Board game mechanics," 2018, *Accessed 28 jul. 2018.* [Online]. Available: https://boardgamegeek.com/browse/boardgamemechanic

[27] R. Hunicke, M. LeBlanc, and R. Zubek, "Mda: A formal approach to game design and game research," in *Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence*, 2004, pp. 1–5.

[28] A. Schmitt, G. Rock, and C. Bettinger, "Glencoe – a tool for specification, visualization and formal analysis of product lines," in *Proceedings of the 25th ISPE Inc. International Conference on Transdisciplinary Engineering*, 2018, pp. 665–673.

[29] K. Czarnecki, "Overview of generative software development," in *Unconventional Programming Paradigms*. Springer, 2005, pp. 326–341.

[30] M. Sicart, "Defining game mechanics," *Game Studies*, vol. 8, no. 2, 2008.

[31] H. Heitkötter, T. A. Majchrzak, and H. Kuchen, "Cross-platform model-driven development of mobile applications with md 2," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 526–533.

[32] M. Palmieri, I. Singh, and A. Cicchetti, "Comparison of cross-platform mobile development tools," in *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*. IEEE, 2012, pp. 179–186.

[33] BoardGameGeek, "Pachisi | board game," 2018, *Accessed 20 jan. 2019.* [Online]. Available: https://www.boardgamegeek.com/boardgame/2136/pachisi

[34] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Computing Surveys (CSUR)*, vol. 28, no. 2, pp. 415–435, 1996.