PCG-RA (Procedural Content Generation Reference Architecture)

(Tempo de leitura: 30 minutos)

Introdução

Arquiteturas de referência são um tipo de arquitetura de software que busca fornecer orientação para o desenvolvimento, padronização e evolução de arquiteturas de sistemas de um domínio definido.

Este documento tem como objetivo especificar o design da arquitetura de referência nomeada de PCGRA (Procedural Content Generation Reference Architecture). A arquitetura proposta busca estabelecer um padrão para jogos que utilizem a geração procedural de conteúdo, permitindo aos jogos adaptar, validar e orquestrar diversos tipos de conteúdos, com um alto nível de controle sobre o processo de geração.

A arquitetura foi desenvolvida com o foco na orquestração de mapas e quests. Apesar disso, pode ser utilizada também para a integração de outros tipos de conteúdos conforme as exigências do jogo. Também existem mecanismos que permitem informar ao sistema de geração os dados do jogador e do jogo a fim de possibilitar a geração do conteúdo de forma adaptada. Além disso, a arquitetura também permite que os conteúdos sejam validados de forma conjunta, a fim de estabelecer um nível mínimo de factibilidade do conteúdo gerado.

Acompanhado com este material, um vídeo foi criado para acompanhar a seção Visões arquiteturais, disponível em apresentacao-pcgra.mkv.

As seguintes seções irão apresentar os objetivos, requisitos, atributos, riscos, stakeholders, e as perspectivas da arquitetura, a fim de possibilitar a sua implementação.

Objetivos

A arquitetura PCG-RA tem os seguintes objetivos definidos:

- **Geração de conteúdos de tamanhos arbitrários e em tempo de jogo**: Muitas vezes, jogos demandam que conteúdos sejam gerados de maneira contínua. Um exemplo desses é o jogo Minecraft que utiliza geração procedural para criar um mundo “infinito”. Para esses tipos de jogos, a arquitetura deve permitir que novos conteúdos sejam gerados conforme demandados, e também deve ser feita de maneira performática, a fim do processo de geração ser executado em jogo.

- **Permitir que novos algoritmos de geração sejam acoplados, substituídos ou modificados**, a fim de permitir que jogos que dependem fortemente da geração de conteúdos tenham a liberdade de explorar novos algoritmos de geração de forma modular.

- **Adaptação dos conteúdos aos jogadores**. Nesse sentido, a geração de conteúdo pode ser aproveitada para gerar conteúdos adaptados à experiência dos jogadores. A arquitetura deve permitir que informações dos jogadores auxiliem na geração de conteúdo.

- **Geração de conteúdo seja fortemente influenciada pelas decisões do designer**: O sistema de geração deve permitir ser influenciado com a alteração de métricas utilizadas para a geração, assim como ser utilizado em conjunto com conteúdos feitos de forma manual, introduzidos pelo designer.

Stakeholders

- **Jogadores**: São os principais usuários que irão experienciar o conteúdo do jogo. Podem ser o usuário final do sistema, um testador ou o próprio designer, que testam o conteúdo antes de disponibilizar aos usuários finais.
- **Designers**: Responsáveis por guiar o processo de criação do conteúdo, podendo alterar os parâmetros utilizados para a geração do jogo e também pode introduzir conteúdos manuais no processo de geração.
- **Desenvolvedores**: Principais responsáveis por dar manutenção no sistema. Atuam em conjunto com os designers para implementar os algoritmos de geração.

Requisitos Funcionais e Métricas de qualidade

As métricas de qualidade são essenciais para definir os requisitos arquitetônicos que servem de base para a criação da arquitetura. As métricas de qualidade definem quais são as características que a arquitetura deve buscar atender.

Diversos estudos acadêmicos foram analisados em busca de definir os requisitos da arquitetura sobre arquitetura no contexto de geração procedural de conteúdo. Ao todo 9 estudos foram analisados, e deles extraídos os principais métricas de qualidade:

Métricas de qualidade	
Modularidade	O quanto o sistema pode ser dividido em partes menores. É influenciado diretamente pelo acoplamento e coesão da arquitetura.
Extensibilidade	O quanto o sistema pode ser estendido para adicionar novos geradores.
Modificabilidade	A facilidade do sistema em ser modificado.
Manutenibilidade	O quanto o sistema pode ser mantido ao longo do tempo.
Portabilidade	A facilidade de portar o sistema para diversos ambientes. Em contexto, para jogos já em desenvolvimento, e para engines de jogos.
Escalabilidade	A facilidade do sistema em aumentar a sua capacidade de geração de conteúdo.
Performance	O quão rápido o sistema é em gerar conteúdo.
Usabilidade	O quanto o sistema se adequa para ser utilizado pelo designer para gerar conteúdo.
Compreensividade	A facilidade do sistema em ser compreendido pelo usuário.
Controlabilidade	O quanto o sistema permite que o usuário tenha o controle sobre os processos de geração de conteúdo.
Factibilidade	O quanto o conteúdo gerado é adequado para o jogo.
Adaptabilidade	O quanto o conteúdo gerado é adaptado para um jogador específico.
Variabilidade	O quanto o conteúdo gerado é variado.

A partir das métricas de qualidade e requisitos extraídos dos estudos, a arquitetura proposta visa atender os seguintes requisitos arquiteturais:

Requisito	Justificativa
Adaptar conteúdos	A fim de permitir gerar conteúdos adequados às narrativas do jogo e a diferentes jogadores.
Colher dados dos jogadores	A fim de adaptar os conteúdos gerados. É necessário adquirir as informações dos jogadores seja durante o jogo ou com questionários.
Colher dados do jogo	A fim de adaptar os conteúdos conforme a progressão do jogo.
Validar/Testar conteúdos	A fim de permitir que os conteúdos sejam validados antes de serem devolvidos aos usuários. Isso pode ser feito para garantir um certo nível de factibilidade ou variabilidade do conteúdo.
Adicionar ou modificar módulos geradores	A fim de permitir a adição de novos geradores de forma modular, w
Modificar parâmetros dos módulos	A fim de permitir gerar conteúdos de tamanhos arbitrários, de forma coerente e a diferentes níveis de granularidade, além de permitir um maior nível de controlabilidade pelo designer.
Orquestrar conteúdos	Para jogos que exijam a geração de mais de um tipo de conteúdo de forma combinada.
Gerar conteúdos sob demanda em tempo real	Muitos jogos desejam gerar os conteúdos sob demanda durante a execução do jogo.
Permitir geração de forma paralela	Uma forma de permitir a geração de conteúdo de forma escalável a nível de arquitetura é permitindo que conteúdos sejam gerados de forma paralela.
Cache de conteúdos gerados	A performance pode ser melhorada com a utilização de um cache de geração, para que os conteúdos não sejam gerados de forma redundante.
Alterar conteúdos manualmente	A fim de permitir ao designer ajustar os níveis sobre sua visão.
Mostrar informações da geração	A fim de fornecer o feedback sobre os efeitos das decisões dos designers sobre o processo de geração.
Interface genérica para o sistema de geração	Uma interface genérica deve permitir que o sistema seja utilizado de forma mais acessível, sem repassar a complexidade interna para o usuário. Além disso, deve permitir que o sistema seja portado mais facilmente para outras plataformas.

Com estes requisitos considerados, a próxima seção irá apresentar as visões arquiteturais que representam a arquitetura.

Visões arquiteturais

As seções a seguir apresentaram a arquitetura mediada por diversos pontos de vista:

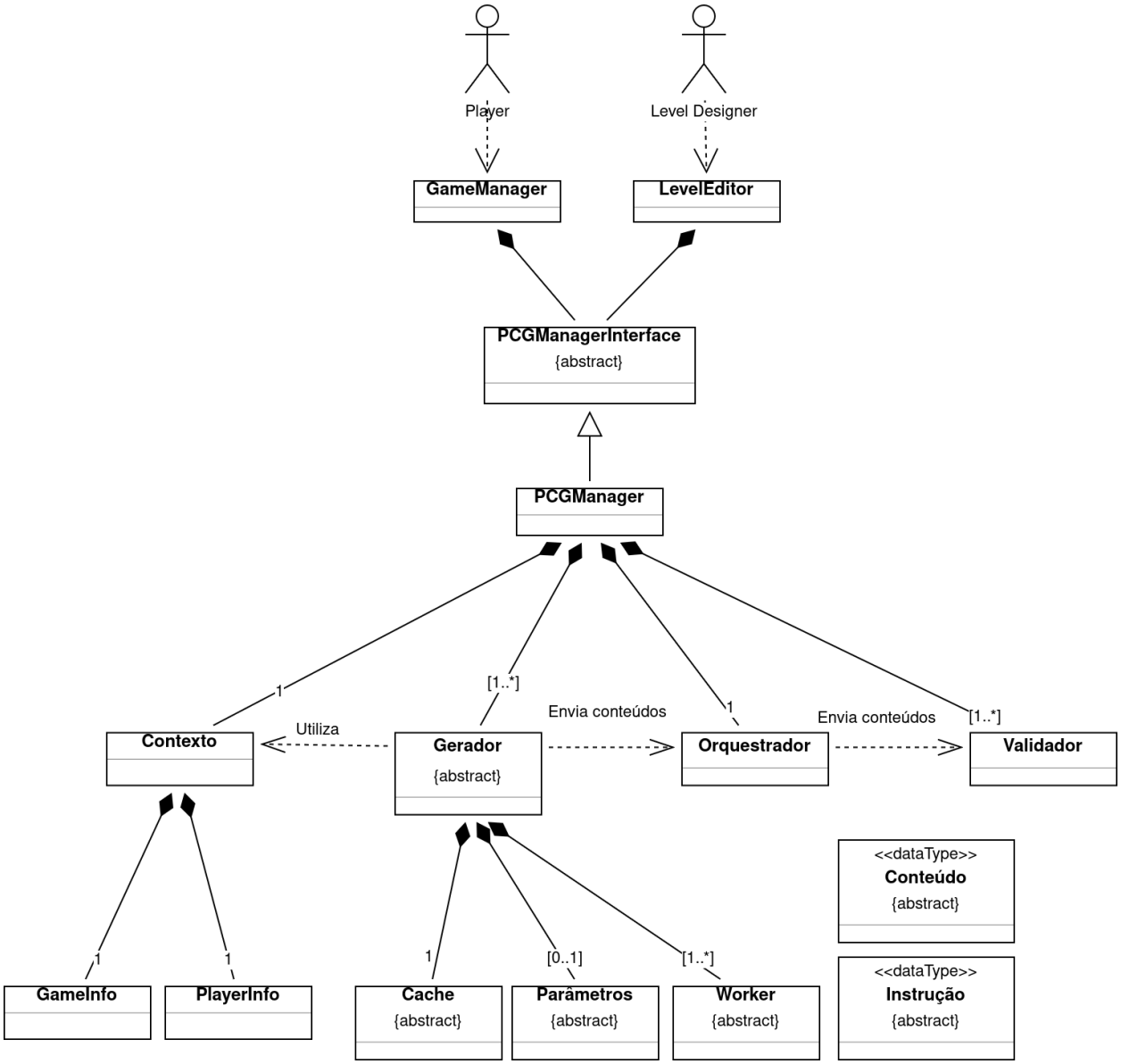
- **Conceitual**: apresenta uma visão de alto nível sobre o sistema. Para este foram utilizados os diagramas de bloco do *SysML*, que permitem descrever os atributos presentes em cada módulo e seu relacionamento com os outros módulos
- **Funcional**: apresenta o sistema do ponto de vista de quais funcionalidades são oferecidas para cada stakeholder;
- **Estrutural**: apresenta a estrutura do código em termos de classes e módulos;
- **Processual**: apresenta a forma da arquitetura em termos de unidades concorrentes. Pode descrever diferentes processos, threads programas ou módulos executados de forma concorrente;

Os diagramas foram elaborados utilizando a ferramenta *diagrams.net*, seguindo os padrões de modelagem *SysML* (*System Modeling Language).*

Adicionalmente um vídeo foi criado para explicar a visão estrutural e o exemplo do gerador de mapas de forma didática. Disponível em apresentacao-pcgra.mkv.

Para melhor compreensão da arquitetura é recomendado iniciar a leitura pela visão conceitual para um panorama geral, seguido da visualização do vídeo, que mostra a funcionalidade da arquitetura de forma didática

Visão Conceitual



Do qual:

- **Conteúdo** é o tipo de dado que será gerado. Pode ser um inimigo, um mapa, uma música, etc.

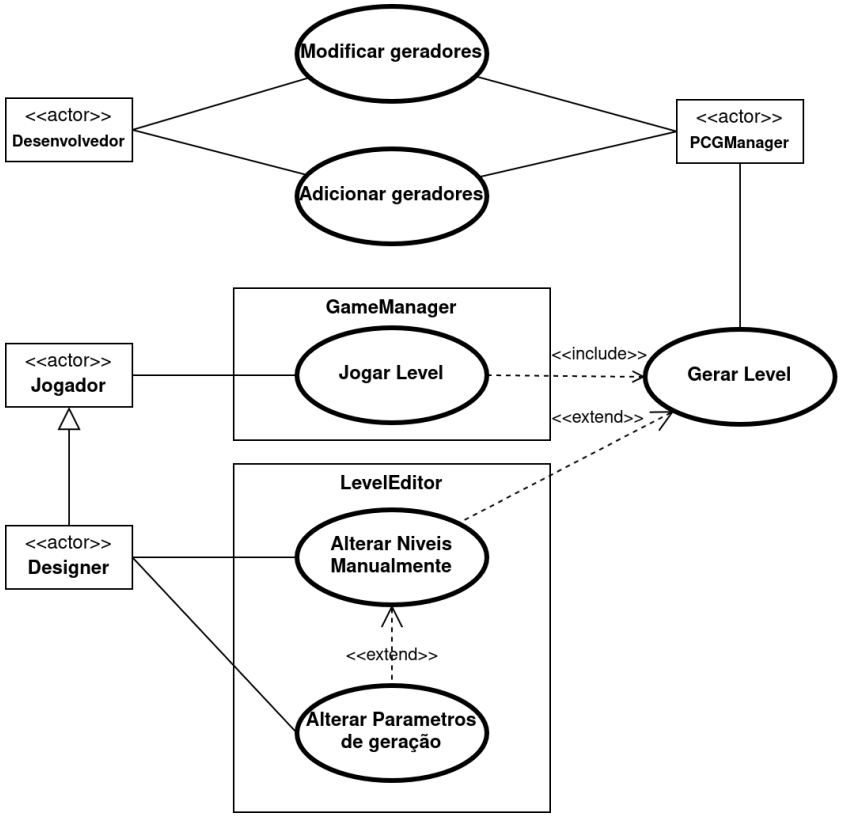
- **GameManager** Implementa a lógica do jogo e apresenta conteúdos gerados ao jogador, requisita a geração de novos conteúdos caso desejado, e adquire informações do jogador caso deseje adaptar os conteúdos;

- **LevelEditor** permite ao designer interagir diretamente com o PCGManager, o designer pode mandar instruções de geração e alterar os parâmetros e contexto de forma direta;

- **Instrução** é o tipo de dado utilizado para solicitar a geração de um conteúdo específico. Deve ser utilizado de acordo com o conteúdo a ser gerado. Em um exemplo de geração de mapa, a instrução pode indicar o quadrante que será gerado.

- **ContentManagerInterface** a interface com o do *PCGManager* com o jogo ou ferramenta. Permite informar os dados do contexto de geração, os parâmetros de geração, requisitar e receber conteúdos;
- **PCGManager** implementa a interface. Controla todo o processo de criação de conteúdo e implementa a interface que interage com os clientes que disponibilizam os conteúdos aos designers e jogadores;
    - **Contexto** Agrega informações contextuais que podem ser utilizadas para auxiliar no processo de geração, a fim de adaptar os conteúdos aos jogadores ou às temáticas específicas do jogo;
        - **GameInfo** é o tipo de dado que agrega Informações do jogo: nível de dificuldade, nível de progresso
        - **PlayerInfo** é o tipo de dado que agrega as Informações do jogador: perfil, respostas de survey
    - **Gerador** controla o processo de geração de um conteúdo específico:
        - **Parâmetros** que são utilizados na geração, por exemplo, em um gerador de mapas os parâmetros podem ser o tipo de terreno a ser gerado, a altura máxima e mínima do terreno;
        - **Cache** utilizado para armazenar os conteúdos já gerados, a fim de não gerar conteúdos redundantes;
        - **Worker** executa uma instrução de geração, pode ser instanciado várias vezes para realizar várias instruções de forma paralela, em que cada worker será inerente a uma thread.
- **Orquestrator** é responsável por juntar todos os conteúdos em artefato único. Apresenta métodos para adicionar os conteúdos individualmente, e um método para combinar todos os conteúdos já adicionados;
- **Validator** valida a factibilidade artefato gerado. Internamente a validação pode ser feita por meio de algoritmos ou agentes inteligentes. Permite a utilização de um limite de factibilidade, que dirá se o conteúdo é adequado ou não;

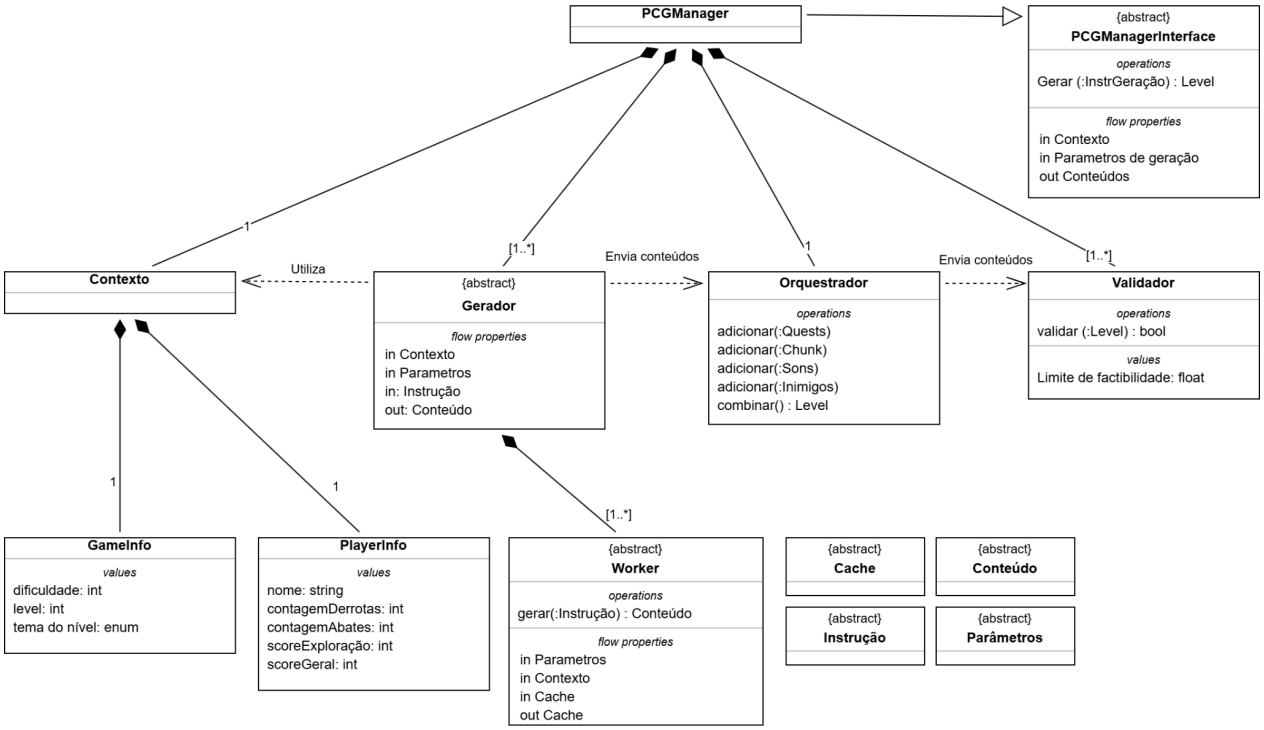
Visão Funcional



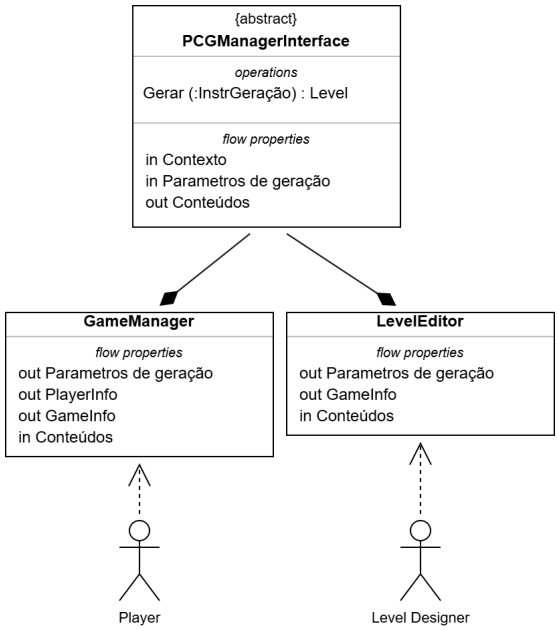
A princípio, os desenvolvedores terão os papéis de modificar e adicionar geradores. Os designers podem alterar níveis de forma manual e alterar os parâmetros de geração. Os jogadores, que são os usuários finais, podem jogar os níveis, gerados com auxílio do designer ou não.

Visão Estrutural

Parte 1



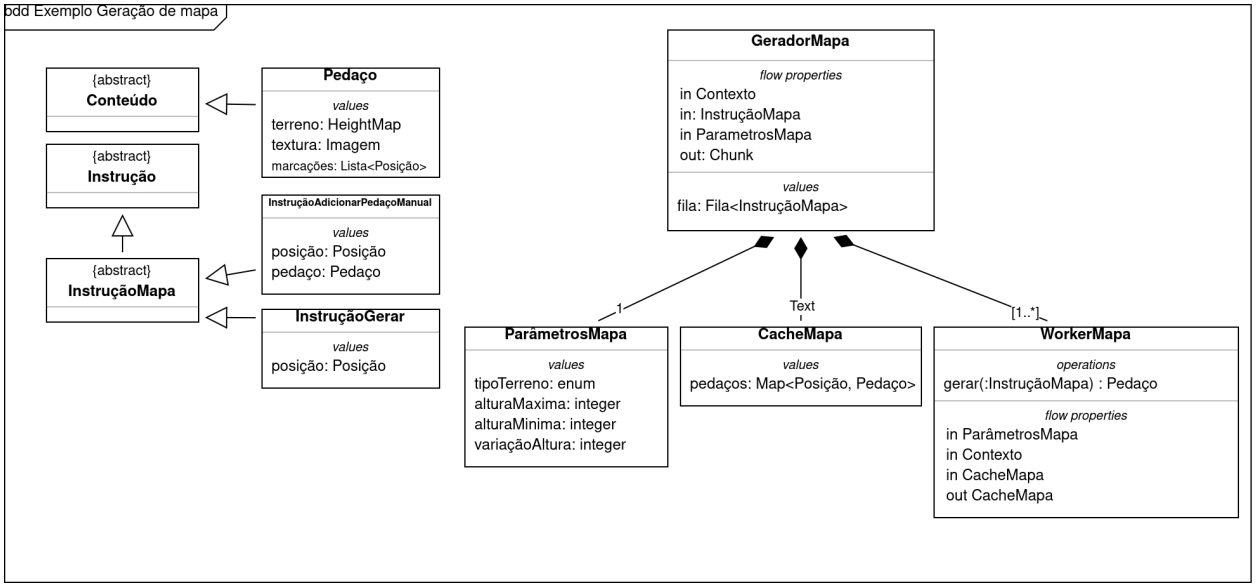
Parte 2



O funcionamento do sistema pode ser observado acima. Como exemplo, os componentes GameInfo e PlayerInfo tem atributos como o nível de dificuldade, level, tema do nível, e o nome, defeitos, abates, e score do jogador.

O Componente *Gerador* recebe continuamente dados do contexto, parâmetros e instruções de geração e utiliza essas informações para gerar os conteúdos a partir do *Worker*.
Conforme os conteúdos são gerados eles são adicionados ao orquestrador, que irá combiná-los e adicionar ao *Validador.* O papel do validador é avaliar o conteúdo e gerar uma pontuação. Esta pontuação será comparada com o limite de factibilidade que, caso não superado, deve reiniciar o processo de geração.
O *GameManager* e *LeveEditor* interagem com a interface do *PCGManager* adicionando continuamente informações do *player* e instruções do *designer.* Desta forma, servem como meio para disponibilizar os conteúdos gerados pelo PCGManager. O *LevelEditor* tem uma interação mais direta com o *designer*, pois permite-lhe um controle mais direto sobre o processo de geração. O *GameManager* por outro lado, deve apresentar a lógica do jogo, logo o jogador interage de forma indireta com o sistema de geração, mediado pela jogabilidade.
A próxima visão estrutural apresenta um exemplo de como um gerador de mapas poderia ser implementado utilizando esta arquitetura.

Visão Estrutural (Exemplo Gerador de Mapas)

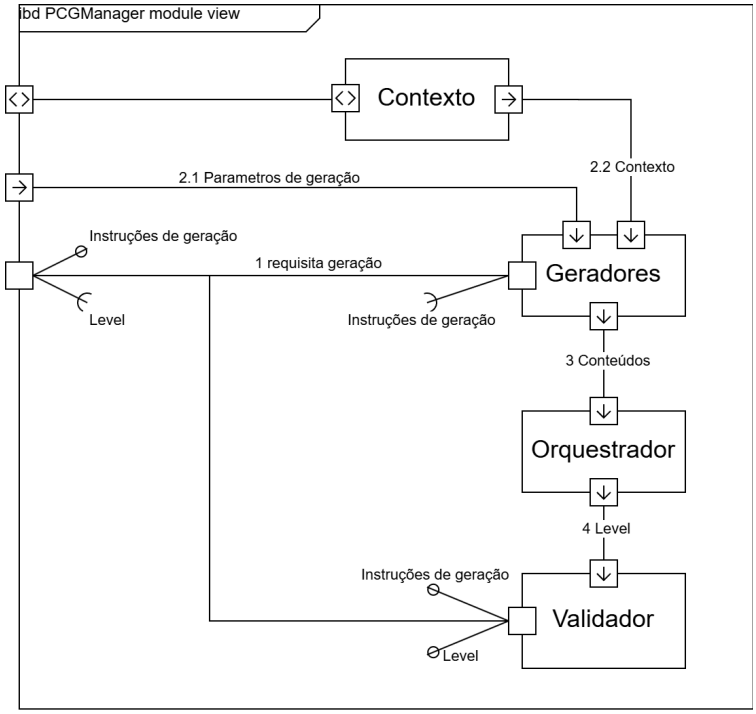


O exemplo acima apresenta uma forma de implementar um gerador de mapas capaz de gerar o mapa de forma modular, em cada “pedaço” do mapa é gerado individualmente. Os componentes são:

- **Pedaço** é o tipo de dado abstraído do tipo *Conteúdo, s*erá o produto final de uma iteração de geração. Ele é constituído de um mapa de alturas (*HeightMap*) que representa a topologia da posição gerada, textura e marcações, que são pontos interessantes para colocar inimigos, tesouros, etc.
- **InstruçãoGerar** é uma instrução para gerar uma posição específica do mapa;
- **InstruçãoAdicionarPedaçoManual** é uma instrução utilizada pelo designer para alterar uma parte do mapa manualmente, o que lhe permite fazer pequenos ajustes, ou até adicionar conteúdos de forma manual, como edifícios específicos do jogo, por exemplo.

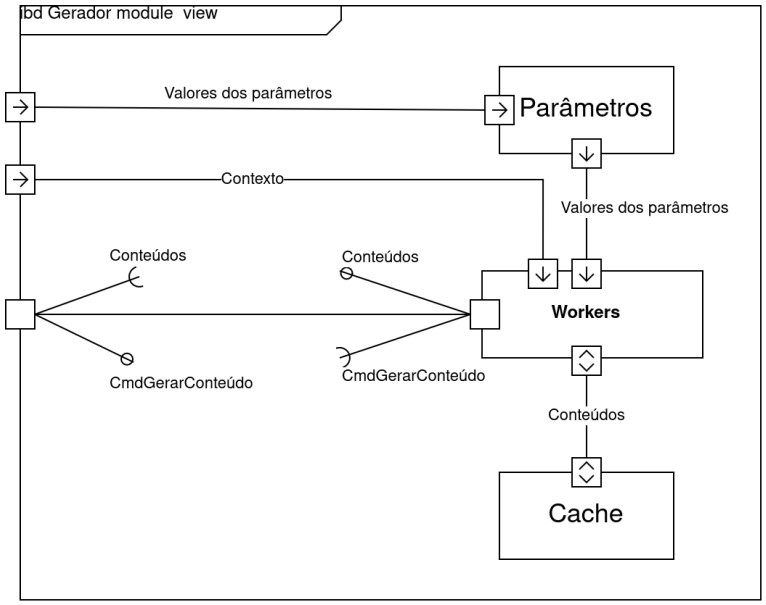
- **ParâmetrosMapa** são o conjunto de parâmetros utilizados para geração. Conforme o exemplo, é constituído dos seguintes atributos:
    - **tipoTerreno**: apresenta o tipo de terreno a ser gerado. Por exemplo, os tipos poderiam ser: “floresta”, “montanhoso”, “riacho”, “glacial”, etc;
    - **alturaMaxima** e **alturaMinima**: influência nos limites da topologia que vai ser gerada;
    - **variaçãoAltura**: altera a uniformidade da topologia gerada. Quanto maior, maior será a diferença de altura entre um ponto e outro.
- **CacheMapa:** armazena os pedaços já criados, caso a mesma instrução seja requisitada novamente, o cache é utilizado para retornar o pedaço armazenado;
- **WorkerMapa:** implementa os métodos de geração do mapa, que utilizam o conjunto do *contexto*, *parâmetrosMapa* e *instrucãoGerar* para gerar um *Pedaço* do mapa. Observe que o *contexto* e *parâmetrosMapa* são adicionados antes da geração ocorrer, o processo de geração só acontece após a chamada do método *gerar*. Pode ser instanciada múltiplas vezes a fim de realizar a geração de forma paralela.

Visão Processual (PCGManager e Gerador)



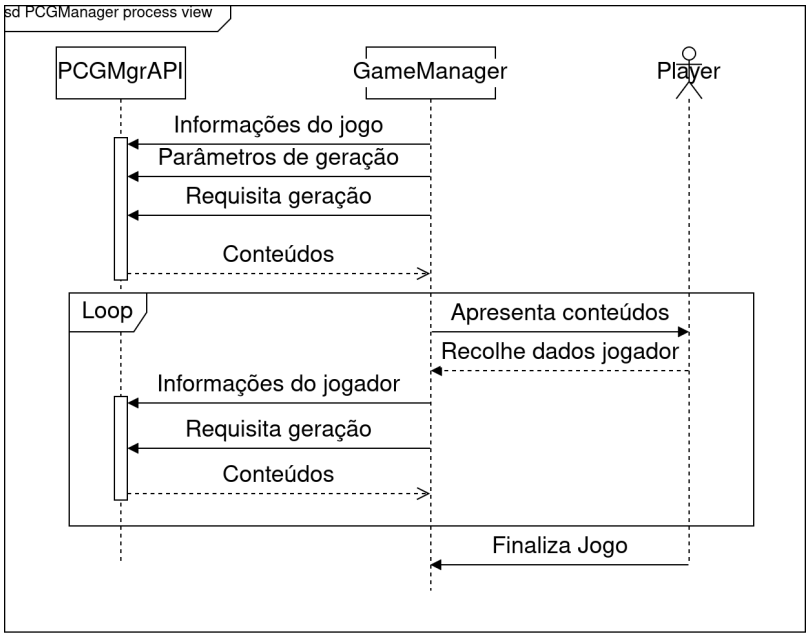
O processo de geração é iniciado quando os geradores recebem uma instrução de geração, os geradores então utilizam dados do contexto e parâmetros recebidos anteriormente, para auxiliar no processo de geração. Após a geração, os conteúdos são repassados para o *Orquestrador* e em seguida para o *Validador*. O *Validador* valida o level gerado e pode requisitar uma nova geração, ou devolver o conteúdo ao cliente.

Visão Processual (Gerador)



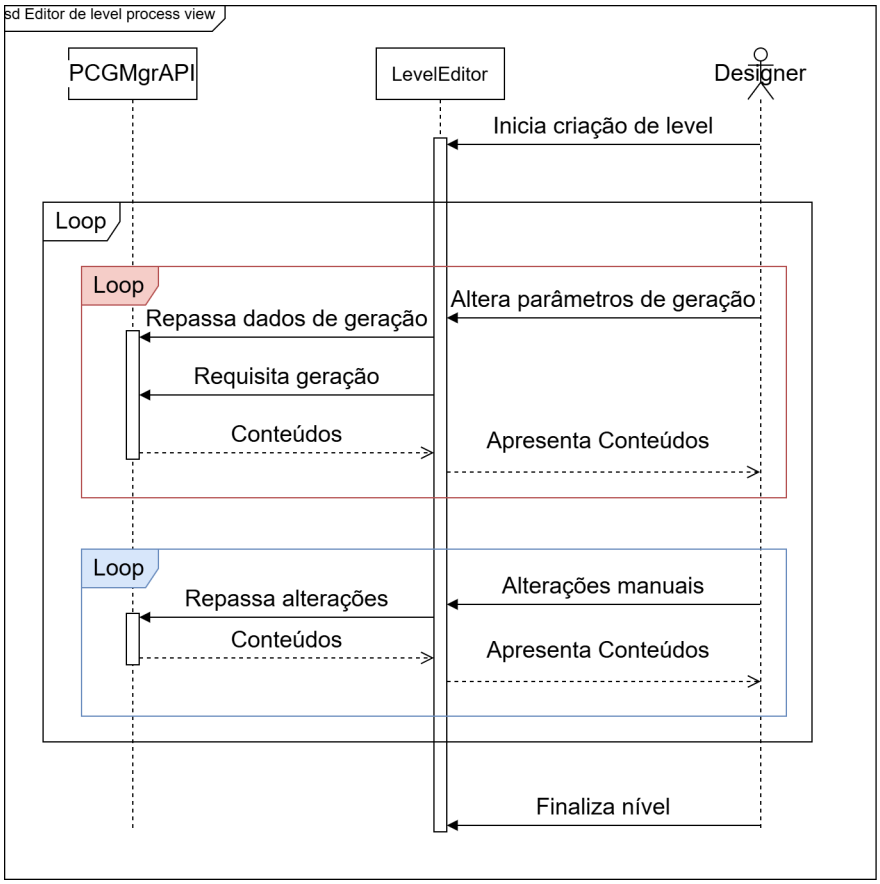
Dentro do gerador, ele é constituído pelos *Parâmetros*, *Workers* e *Cache.* O processo de geração é realizado quando o *Worker* recebe o comando para gerar o conteúdo, quando ele consulta os *Parâmetros, Contexto* e *Cache* para realizar a geração.

Visão Processual (GameManager)



A interação entre o jogador e o *PCGManager*. As informações do jogo e parâmetros de geração são fornecidos pelo *GameManager* para o *PCGManager* antes do jogo iniciar, a primeira interação com o jogador é o recebimento dos conteúdos. Interativamente os conteúdos são apresentados ao jogador, enquanto o jogador desfruta do jogo, informações dele são adquiridas e utilizadas para gerar conteúdos posteriores.

Visão Processual (LevelEditor)



Diferentemente do *GameManager*, no processo de criação do nível o designer tem um controle muito maior sobre os processos de geração. O designer tem duas interações principais com o *PCGManager* (mediadas pelo *LevelEditor*): (1) Alterar os parâmetros de geração e (2) alterar os conteúdos manualmente.

Riscos

Destacam-se alguns riscos na utilização do *PCGRA*:

- Degradação da arquitetura, quando a qualidade da arquitetura se deteriora ao longo do tempo;
- A arquitetura pode não abranger todo o escopo da geração, caso uma técnica de geração desejada não possa ser implementada dentro dela;
- O módulo de validação pode ser desnecessário caso nenhuma técnica de validação seja aplicada;
- Muitas vezes múltiplos conteúdos são gerados de forma conjunta, neste caso o componente de orquestração de conteúdo se torna irrelevante. Uma vez que o conteúdo já é criado de forma conjunta;