

**Faculdade de Engenharia da Universidade do Porto**



**ESOF 2016/2017 - Mestrado Integrado em Engenharia Informática e Computação**

**Autores:**

Andreia Rodrigues - [up201404691@fe.up.pt](mailto:up201404691@fe.up.pt)

Eduardo Leite - [gei12068@fe.up.pt](mailto:gei12068@fe.up.pt)

Gonçalo Leão - [up201406036@fe.up.pt](mailto:up201406036@fe.up.pt)

Francisco Queirós - [up201404326@fe.up.pt](mailto:up201404326@fe.up.pt)

ESOF - Relatório 1

WorldEdit

Descrição do Projecto

O WorldEdit é um *plugin* para o jogo Minecraft que permite editar o mundo 3D do jogo através de funcionalidades novas fornecidas por este *plugin*.

O Minecraft é um jogo do tipo *sandbox* e *openworld* onde o jogador pode construir qualquer coisa num mundo 3D gerado aleatoriamente a partir de blocos/cubos de diferentes materiais (voxels). Os jogadores podem ainda explorar o mundo, recolher recursos e combater contra inimigos ou outros jogadores.

Este pode ser jogado tanto no modo *singleplayer* como em *multiplayer e*  suporta *plugins*, uma extensão do servidor que adiciona novas funcionalidades ao jogo ou modifica as já existentes sem que seja necessário um *client custom* para aceder ao mesmo (o que o diferencia de um *mod*).

O projeto que vamos analisar é *open source* e está sob a licença GNU Lesser General Public License v3.

O objetivo deste *plugin* é fornecer um modo mais rápido e eficiente de editar o mundo do jogo.

Das funcionalidades de edição do mundo, podem-se destacar:

· Criar construções de uma forma mais rápida;

· Criar, substituir ou apagar centenas de blocos em segundos;

· Nivelar terreno.

· Usar novas *tools* e *brushes* para construir montanhas de blocos;

· Gerar esferas, cilindros, cubóides, etc;

· Teletransportar o jogador para outras áreas apenas clicando ou usando um comando na consola;

· Escolher uma área e restaurar o seu estado anterior, através de *backups*.

O projeto foi inicializado em Setembro de 2010 por Albert Pham (“sk89q”), que juntamente com outros três utilizadores (“TomyLobo”, “wizjany” e “ zml2008”), tiveram o maior impacto no desenvolvimento do projeto.

Foi lançada a primeira versão “WorldEdit 4.6” em Agosto de 2011. Até agora, este *plugin* conta com mais 15 milhões de downloads, sendo a versão “WorldEdit 6.1” a mais bem-sucedida, lançada em 2015, com 2 milhões de *downloads*.

A principal linguagem de programação usada é Java, que no momento da elaboração deste relatório, engloba cerca de 99,2% segundo o Github. Os 0.8% restantes do código estão escritos usando a linguagem de programação Javascript, e consistem em *craftscripts*. Um *craftscript* é um ficheiro que permite a execução, através de um único comando, de uma tarefa complexa de edição do mundo do *Minecraft*, como a geração aleatória de um labirinto.

Tem um total de 64 contribuidores, 8 dos quais estiveram presentes durante todo o desenvolvimento e são os responsáveis pelas contribuições mais relevantes. Embora frequentemente atualizado, já é um projeto bastante completo, por isso não está em grande desenvolvimento de momento.

Processo de Desenvolvimento

Um processo de desenvolvimento de software define um conjunto estruturado de atividades para desenvolver um sistema de software. Seguir um processo de desenvolvimento aumenta a eficiência e consistência do trabalho de desenvolvimento do sistema e permite detetar aspetos a melhorar.

Vamos começar por falar de alguns princípios gerais do desenvolvimento do projeto para o World Edit apontados pelo seu principal contribuidor.

Logo após, vamos brevemente apresentar a comunidade do WorldEdit.

De seguida, vamos ver com mais detalhe como funciona o sistema de contribuições para o projeto.

Subsequentemente, vamos nos debruçar sobre o modelos que vão ao encontro do processo de software adotado.

Por fim, vamos analisar a evolução do projeto e a estrutura do repositório.

Aspetos gerais

Segundo Albert Pham (conhecido por “sk89q” no Github), o principal contribuidor do projeto, o processo de desenvolvimento deste projeto rege-se pelos seguintes princípios:

- pequenas alterações ao código são *commited* diretamente no *branch* principal;

- grandes alterações são *commited* para *branches* individuais e são desenvolvidas ao longo do tempo nesse mesmo *branch* até à funcionalidade estar completa. Quando essa funcionalidade está pronta para ser *merged*, é criado um *pull request* e o código é revisto para proceder ao *merge* com o *branch* principal (*master*), se for aprovado;

- testes unitários são corridos frequentemente, tal como ferramentas que permitem detectar se existe algo suspeito no código (para garantir segurança);

- sugestões de novas funcionalidades e reports de *bugs* são registados no *issue tracker*;

- novas versões do *plugin* são lançadas periodicamente.

Comunidade

A comunidade do WorldEdit é composta pelos colaboradores do projeto e por quem usa este *plugin*. A comunidade tem uma grande facilidade de comunicação com os autores do projeto, sendo frequente sugerirem novas funcionalidades, tirarem dúvidas e fazerem *reports* de *bugs* que encontram no *plugin*.

Para contacto direito com o autor principal do projeto Albert Pham, um dos meios de comunicação disponíveis é um canal IRC, comum a todos os seus projectos (<http://skq.me/irc/irc.esper.net/sk89q/>), e o seu twitter pessoal(<http://twitter.com/sk89q>).

Existe também um fórum (<http://forum.enginehub.org/>), onde os membros da comunidade comunicam regularmente entre si e com os colaboradores do projeto sobre assuntos diversos, sendo os mais comuns dúvidas acerca de funcionalidades do plugin e report de erros.

Existe ainda um *issue tracker* (<http://dev.enginehub.org/youtrack/issues/WORLDEDIT?p=0&f=false>), onde grande parte dos *reports* são sobre erros e atualizações do plugin, ou tratam-se de propostas para novas funcionalidades. Cada issue é acompanhado de informação acerca da plataforma em questão (se alguma) e do seu estado, por exemplo, não serem capazes de reproduzir novamente um bug, já terem resolvido o problema, etc.

Por fim, existe também o site oficial do plugin (http://www.enginehub.org/worldedit) onde podemos encontrar informações acerca deste, videos de como o utilizar, como fazer download e instalação, questões frequentes, documentação do plugin, fórum de discussão e issue tracker já mencionados e o código fonte do projeto.

Contribuições

Qualquer membro da comunidade pode contribuir para este projeto. No entanto, as regras que os autores do projeto definem são bastante rígidas. Estas regras estão definidas num ficheiro da pasta principal, CONTRIBUTING.md, que por sua vez é referido no README.md do projeto.

Um exemplo notável de uma regra a seguir para que uma contribuição seja aceite é que o novo código siga as convenções de programação da Oracle. A título de exemplo, uma convenção recomendada para programação em Java (mas que pode ser estendida para outras linguagens semelhantes) é de declarar apenas uma variável em cada linha de código (isto incentiva a escrita de comentários a explicar o papel dessa variável no programa).

Outra regra que deve ser seguida pelos contribuidores é que o código deve ter sido devidamente testado.

Depois de feito um *pull request*, o código é testado novamente na totalidade com testes unitários por parte de outros membros do projeto, antes de ser aceite e *merged* com o *branch* principal.

Modelos de processo de software usados

Um modelo de processo de *software* é uma representação abstrata de um processo de desenvolvimento de *software*.

O projeto rege-se segundo um modelo próximo do “Incremental Development and Delivery” que consiste em desenvolver o projeto incrementalmente e iterativamente, e avaliar cada incremento antes de proceder para a próxima tarefa. Cada incremento consiste no desenvolvimento de uma ou mais *features* ou na resolução de um ou vários *bugs*, reportados através do *issue tracker* do projeto.

Tal como referido na secção dos “Princípios gerais” do desenvolvimento do software deste projeto, várias funcionalidades novas foram sendo desenvolvidas ao mesmo tempo em *branches* diferentes, segundo o modelo “Waterfall”, onde a funcionalidades são definidas, é feito o *design* do software, implementado o código, são feitos testes unitários, é feita a integração e testes no sistema, e por fim é colocada a funcionalidade do *branch* principal.

O projeto também segue de perto vários princípios do desenvolvimento ágil (*Agile*) de software pois o software encontra-se em constante evolução (*commits* ocorrem a um ritmo semanal, nos períodos mais ativos), incorporando novas *features* e corrigindo *bugs*, graças a uma rede de dezenas de colaboradores que cooperam entre si e estabelecem diálogo a um ritmo frequente através de discussões em cada tópico do *issue tracker* ou de cada *pull request*, por exemplo.

Evolução do projeto

O projeto teve mais atividade nos primeiros 2 anos de desenvolvimento do *plugin*, entre 2010 e 2011, e voltou a ter grande atividade em 2014.

De momento, o projeto não se encontra muito ativo, tendo sido já intensivamente desenvolvido no passado e portanto trata-se um projeto bastante completo.

No momento da elaboração deste relatório, o *commit* mais recente para o *master branch* é do dia 2 de Setembro de 2016, e os últimos *commits* relacionam-se com pequenas correções do código e atualizações para a plataforma Forge, uma API de *modding*, que torna mais fácil criar novos *mods* e verificar se são compatíveis com *mods* já existentes.

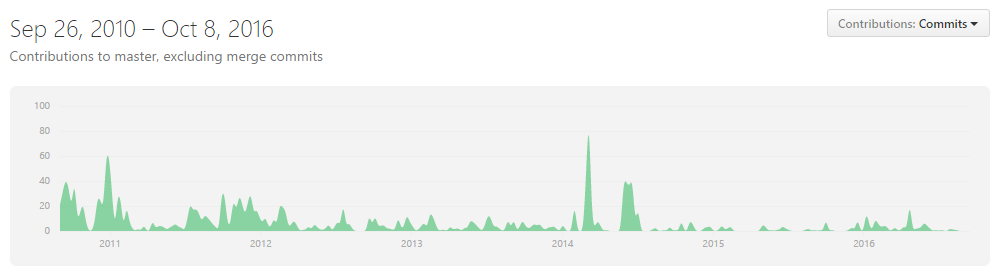


Figura 1: Evolução dos commits do projeto ao longo dos anos.

<https://gyazo.com/186189a725bf4c6d6142a645bf1ca704>

Organização do repositório

Existem um total de 27 *branches*, relativas a funcionalidades que estão a ser desenvolvidas à parte. Após o desenvolvimento e posterior análise do código, o *branch* é *merged* com o *branch* principal.

Existem também um total de 103 *releases*, correspondentes a lançamentos *alpha*, *beta* e oficiais do *plugin*. As descrições detalhadas de todas as funcionalidades adicionadas em cada lançamento estão registadas num ficheiro “CHANGELOG.txt” no *branch* principal.

Análise Crítica

Vamos analisar o projeto WorldEdit segundo várias perspetivas, para tentar de alguma forma medir a sua qualidade.

Começamos por avaliar as contribuições para o projeto, desde a sua frequência e pertinência.

De seguida, analisar-se-á a organização do projeto, tanto ao nível do código, como do repositório.

Por fim, discutir-se-á o processo de software usado. Em particular, discutir-se-á porque é que é pertinente que o processo se aproxime do modelo incremental, e explorar-se-á eventuais alternativas, salientando os seus prós e contras.

Contribuições

* Frequência

Tal como mencionado na secção “Evolução do projeto” em “Processo de Desenvolvimento”, o projeto encontra-se agora numa fase menos ativa da sua existência. Nos últimos dois anos, foi havendo alguma atividade com a adição de pequenas novas funcionalidades e correção de *bugs*, muitas das vezes resultado do *feedback* da comunidade de jogadores. Esta atividade mostra que o projeto continua a ser relevante para a comunidade de jogadores de *Minecraft*.

Nos últimos anos, registaram-se dois grande períodos de atividade.

Por um lado, a grande atividade durante os anos de 2010 e 2011, os dois primeiros do projeto, mostra que a ideia subjacente ao projeto (permitir editar mundos do *Minecraft* de uma forma mais simples e eficaz) foi de grande interesse para a comunidade do *Minecraft*.

Por outro lado, a época mais ativa de 2014 foi devido ao ganho de notoriedade da plataforma Spigot, que permitiu integrar *plugins* no jogo *Minecraft* mais facilmente.

* Pertinência

A maioria dos *commits* vêm acompanhados de uma mensagem elucidativa sobre o que foi alterado ou adicionado ao projeto.

No gráfico de *Code Frequency* no *Github*, podemos constatar que, na maioria das semanas, o número de adições de linhas de código é próximo do número de eliminações. Isto deve-se ao facto que alterar uma linha de código é contabilizado pelo *Git* como uma eliminação seguida de uma adição. Assim, estas observações são coerentes com o facto que a maioria dos *commits* referem-se a correções de erros. Pelo que se pôde averiguar, outra parte significativa dos *commits* correspondem a adaptações para novas versões do *Minecraft*. Logo, podemos concluir que o conteúdo de uma grande parte das contribuições foram relevantes para a evolução do projeto.

Um fator que motiva a pertinência das contribuições é o conjunto de regras para que uma contribuição seja aceite pelos restantes colaboradores do projeto. Estas regras são referidas no README.md do projeto e portanto são fáceis de encontrar por novos colaboradores que desejem contribuir para o projeto.

Na nossa opinião, as regras que foram definidas fazem sentido. Voltemos aos dois exemplos que foram dados na secção sobre “Contribuições” no “Processo de Desenvolvimento”.

Por um lado, seguir uma convenção (neste caso, as convenções de programação da Oracle) traz a vantagem de aumentar a legibilidade do código pois outros contribuidores que seguirem essas mesma regras perceberão o código melhor e de forma mais rápida.

Por outro lado, a norma (trivial) que o código tenha de ser devidamente testado antes de ser submetido permite minimizar a quantidade de *bugs* que são introduzidos no *software*.

Organização

* Código

Tal como fora mencionado antes, o código segue um conjunto de normas específicas (como as “Oracle coding conventions”), o que aumenta a legibilidade do código (por todos os contribuidores adotarem essas convenções) e evita o que os contribuidores principais entendem ser más práticas de programação.

A estrutura dos pacotes do código (*packages*) está muito bem conseguida, o que diminui bastante as desvantagens trazidas pelo modelo de desenvolvimento incremental.

Por exemplo, uma desvantagem do modelo de desenvolvimento incremental é que a estrutura da aplicação tem tendência a degradar-se à medida que vão sendo feitas incrementos. Contudo, dado que o projecto está dividido em *core/bukkit/forge/sponge*, o projecto nunca fica demasiado grande, pelo que acabam por não sofrer tanto com este efeito.

Uma outra vantagem, que está ligada à primeira, é o facto que a existência de um *core* (conjunto de código comum a todas as plataformas) também ajuda a superar as dificuldades, reduzindo os problemas de código duplicado característicos deste tipo de estratégia.

* Repositório

***Branches***

Analisando o repositório podemos concluir que é utilizado o “git branching model”, um modelo adequado ao projeto em causa. Usa uma estrutura de *branches* que permite o desenvolvimento paralelo de diferentes funcionalidades de uma forma mais independente, organizada e eficaz. Estes *branches* são *merged* com o *branch* principal quando a funcionalidade tiver sido completamente desenvolvida e o seu código tiver sido testado intensivamente pelos autores do projeto.

**Documentação (do projeto)**

Na nossa opinião, o projeto encontra-se bem documentado. A pasta principal do projeto contém ficheiros .md (como README.md, CONTRIBUTING.md...) com informações que se encontram bem organizados e são claros nas mensagens que pretendem transmitir, o que torna o projeto mais fácil de compreender e mais convidativo para quem quiser contribuir.

Processo de software

* Modelo usado

Tal como se referiu anteriormente, acreditamos que o modelo adoptado pelos desenvolvedores deste projeto segue mais de perto o modelo *Incremental Development and Delivery*.

Nós acreditamos que este modelo é o mais adequado tendo em conta a constante atualização do jogo Minecraft com novas versões e as atualizações que o plugin tem de ser sujeito para que se mantenha a par das novas funcionalidades requeridas pela comunidade do jogo. Além disso, este modelo torna mais fácil para novos membros contribuírem para o projeto dado que é mais intuitivo.

* Comparação com o modelo *Waterfall*

O modelo de desenvolvimento de software *Waterfall* não é compatível com ambientes de desenvolvimento como no caso do nosso projeto, *WorldEdit*, que é um ambiente onde o controlo de versão é aberto ao público e existe um ritmo alto de lançamento de novas versões.

Com efeito, este modelo não seria compatível com o projeto em estudo pois os seus requisitos estão em constante evolução, à medida que os contribuidores vão propondo novas funcionalidades.

Além disso, seguir o modelo em cascata à risca impediria a sobreposição de atividades de levantamento de requisitos, implementação de *features* e teste do código. Assim, enquanto um colaborador estivesse a implementar uma nova função, outros não poderiam estar a testar outros aspetos da aplicação.

* Comparação com o modelo *Spiral*

O modelo de desenvolvimento *Spiral* prevê um componente de cálculo de risco para além da típica prototipagem dos outros métodos ágeis, incluindo o utilizado no projeto.

Esta componente de cálculo de risco não é necessária no contexto do produto do projeto visto que este é oferecido como uma ferramenta útil e de livre acesso, sem ganhos financeiros ou de reputação por parte dos colaboradores. Por outras palavras, o risco é virtualmente inexistente, pois não há restrições de tempo nem de dinheiro.

Para além da desnecessidade desse paradigma, a implementação é custosa ou morosa, o que é desinteressante num contexto não-profissional.

* Comparação com o modelo *Software Prototyping*

O modelo *Software Prototyping* prevê a existência de um protótipo descartável. Embora o modelo que determinamos como sendo o mais próximo ao utilizado, *Incremental Development and Delivery*, também utilize as perspectivas do modelo *Software Prototyping*, no caso do IDD, o protótipo é uma base sobre qual se itera.

Se *Software Prototyping* fosse o modelo utilizado, para cada funcionalidade ou melhoria que se achasse necessária, criava-se o protótipo para essa alteração como prova de conceito, e se fosse aceite, desenvolver-se-ia o projecto do início com este novo aspecto em consideração. Com efeito, mesmo que o protótipo fosse aprovado, este seria uma reprodução de qualidade inferior à de um produto final. O desenvolvimento da versão final dessa alteração iria criar ainda mais um atraso, para além da reconstrução do projecto.

**Conclusões**

Apesar do *WorldEdit* ser um projecto *open-source* com muitos contribuidores, sentimos que existe um nível substancial de organização para este tipo de projeto, sendo que os principais contribuidores são um grupo que nos parece organizado e experiente.

Os colaboradores foram capazes de garantir que o código fosse testado e de boa qualidade.

Concordamos que o modelo de entrega incremental utilizado (com algumas nuances inevitáveis num contexto prático) era o mais óbvio para este tipo de projecto e achamos que tomaram boas medidas (enunciadas anteriormente) para combater os problemas associados com este tipo de abordagem.

Bibliografia

Um link para um documento com convenções de programação em Java:

<http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>