Universidade Federal de Pernambuco

Centro de Informática

Graduação em Engenharia da Computação

MODELO DE PROCESSO PARA MODIFICAÇÃO DE JOGOS COM CÓDIGO INDISPONÍVEL UTILIZANDO DLLS

PEDRO IVO SILVEIRA SOUSA

Orientador: Geber Lisboa Ramalho

Co-Orientador: Túlio Caraciolo

TRABALHO DE GRADUAÇÃO

Recife, Julho de 2017

Pedro Ivo Silveira Sousa

MODELO DE PROCESSO PARA MODIFICAÇÃO DE JOGOS COM CÓDIGO INDISPONÍVEL UTILIZANDO DLLS

TRABALHO DE CONCLUSÃO DE CURSO APRESENTADO À UNIVERSIDADE FEDERAL DE PERNAMBUCO – UFPE, COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE BACHAREL EM ENGENHARIA DA COMPUTAÇÃO.

Universidade Federal de Pernambuco

Centro de Informática

Recife – Pernambuco

Julho de 2017

Pedro Ivo Silveira Sousa

MODELO DE PROCESSO PARA MODIFICAÇÃO DE JOGOS COM CÓDIGO INDISPONÍVEL UTILIZANDO DLLS

TRABALHO DE CONCLUSÃO DE CURSO APRESENTADO À UNIVERSIDADE FEDERAL DE PERNAMBUCO - UFPE, COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE BACHAREL EM ENGENHARIA DA COMPUTAÇÃO.

BANCA EXAMINADORA:

Orientador: _____

Dr. Geber Lisboa Ramalho (UFPE-CIn)

Avaliador: _____

Dr. André Santos (UFPE-CIn)

Recife, 11 de Julho de 2017.

Dedico este trabalho à minha mãe, que sempre apoiou e incentivou meu crescimento profissional.

Agradecimentos

Agradeço a todos que fizeram parte de forma direta ou indireta deste trabalho de graduação. Em especial, meus agradecimentos a:

Ao meu orientador e ao meu co-orientador, professor Geber Ramalho e Túlio Caraciolo, pela paciência, pelo tempo gasto, pela ajuda e pela confiança depositada na realização deste trabalho de graduação.

Aos meus colegas de trabalho por disponibilizarem seu tempo e experiência para me auxiliar no desenvolvimento deste trabalho de graduação.

Aos meus amigos, André Vitor, Carlos Filipe, Igor Holanda e Renan Spencer por todo apoio e compreensão durante esse longo caminho que tem sido minha graduação.

À Universidade Federal de Pernambuco, em especial ao Centro de Informática, por oferecer todo o suporte e oportunidades necessárias para minha formação.

Aos meus pais, Iraci e Paulo, por todo o apoio, suporte e base que nunca me deixaram faltar para alcançar todos os meus objetivos e sonhos na universidade e na vida.

Resumo

Modelo de processo para modificação de jogos com código indisponível utilizando DLLs

Autor: Pedro Ivo Silveira Sousa

Orientador: Prof. Geber Lisboa Ramalho

Jogos MMO (*Massively Multiplayer Online*) possuem longa vida útil, mas após serem descontinuados por suas empresas são muitas vezes modificados por fãs ou desenvolvedores independentes que, mesmo sem o acesso ao código fonte, buscam atualizar o jogo e manter seu público entretido. Alterar o comportamento de NPCs (Non-Player Characters) para adequá-los às novas necessidades é uma atividade complexa quando não se possui o código fonte. Adaptando processos presentes na literatura, este trabalho de graduação visa propor um modelo de processo genérico que facilita entendimento e reduz o tempo de desenvolvimento na modificação do comportamento de NPCs para jogos MMOs cujo código fonte não está disponível. Estas modificações devem ser realizadas através de inserção de DLLs (*Dynamic-link library*). Para validação do modelo e análise de resultado faremos uma experimentação no jogo Supreme Destiny Asgard, que possui milhares de jogadores diariamente.

Keywords: massively multiplayer online, games, mods, modder, dll, npc, assembly

Abstract

PROCESS MODEL TO MODIFY GAMES WITH UNAVAILABLE SOURCE CODE USING DLLS

Author: Pedro Ivo Silveira Sousa

Advisor: PhD Geber Lisboa Ramalho

MMO games (Massively Multiplayer Online) are long lasting games, but after being discontinued by their publishers, they are sometimes modified by fans or independent developers which even without access to the source code seek to update the game and keep its public. Modifying behaviour of NPCs (Non-Player Character) to suit them to the new necessities is a complex activity without the source code. Adapting processes found in relevant articles this theses seeks to propose a generic process model which will ease the understanding and reduce the time for development of such modifications on NPC behavior for MMO games with no available source code. These modifications must be done through DLL (Dynamic-link library) insertion. To validate the model and analyze the results we'll execute an experiment on the game Supreme Destiny Asgard, which has thousand active daily users.

Keywords: massively multiplayer online games, mods, modder, npc, dll, assembly

Lista de tabelas

Lista de ilustrações

Sumário

1 Introdução

1.1 Motivação

Jogos MMO (*Massively Multiplayer Online*) possuem longa vida útil, mas por não continuarem sendo lucrativos para seus desenvolvedores, ou simplesmente por falta de recurso, são descontinuados para dar lugar a novos jogos. Desenvolvedores independentes ou *publishers* menores ainda podem lucrar e aproveitar do público destes jogos. Para realizar tal feito, eles buscam fazer mudanças em cima do jogo original e criar versões novas ou gerar modificações apenas para aprimorá-lo e atualizar sua jogabilidade **[3; 12; 13]**.

Modificar jogos não é algo novo. Esta tendência surgiu há décadas, porém atualmente tem se tornado cada vez mais convencional na cultura *gamer* **[13; 17]**. Estas modificações são conhecidas como mods (uma abreviação para *modifications*) e são bastante comuns **[3; 17]**. Os *modders*, como são conhecidos aqueles que fazem mods, possuem diversas motivações, desde artísticas à busca por um espaço no mercado **[3]**, isto é motivado pela importância que um mod pode ter para o sucesso de um jogo específico **[3]** trazendo ainda mais imersividade que o original **[12]**.

Um dos exemplos de maior sucesso entre os mods é o *Counter-Strike* (CS), mostrado na **Figura 1**, mod de *Half-Life* (**Figura 2**) pela *Valve Software*. Neste exemplo, milhares de jogadores preferiram jogar o mod ao jogo original, atraindo ainda mais modders e fazendo com que a Valve modificasse seu modelo de negócio e desenvolvimento para inserir mod como parte da experiência dos seus jogadores **[1]**. Em 2001, CS foi o primeiro mod a ser lançado comercialmente, alcançando uma venda totalizada em milhões de dólares **[14].**

Figura 1 - Gameplay de Counter-Strike, Mod de Half-Life. *(Fonte: Google Imagens)*

Figura 2 - Gameplay de Half Life. *(Fonte: Google Imagens)*

Na modificação de jogos de terceiros, programadores frequentemente não possuem acesso ao código fonte **[16]**, e por tanto precisam aplicar técnicas como inserção de DLLs(*Dynamic-link library*) e *hooks* **[11;16]**. Estas técnicas apesar de comuns possuem diversas limitações **[11]**, algo que torna a inserção de novos conteúdos e modificações no comportamento de NPCs (*Non-Playable Characters*) uma atividade muito mais árdua, necessitando um maior tempo para programação e diminuindo a estabilidade do código, algo que afeta diretamente os custos de desenvolvimento.

1.2 Objetivos

Este trabalho de graduação visa propor um processo que facilite o desenvolvimento e reduza a quantidade de erros gerados na modificação do comportamento de NPCs para jogos MMO cujo código fonte não está disponível. Para este processo se faz necessário ter o id dos pacotes utilizados na comunicação entre servidor e cliente. Para alcançar tal objetivo, iremos:

- Criar um processo genérico para modificar o comportamento de um NPC;
- Modelar o processo proposto de maneira gráfica e de fácil entendimento;
- Realizar um estudo de caso utilizando o processo proposto, como experimentação, no executável de um jogo;
- Avaliar o impacto do modelo proposto no tempo de desenvolvimento da solução;

O foco deste trabalho se encontra no primeiro e segundo passo descritos acima, tendo como maior objetivo propor um modelo de processo que facilite e acelere a modificação de comportamento de mobs(NPCs) em jogos MMOs.

1.3 Abordagem

Para alcançar os objetivos traçados, iremos definir alguns comportamentos básicos de mobs e utilizá-los para modelar um processo. Depois, utilizando inserção

de DLLs, realizaremos a experimentação e teste do processo proposto no jogo *Supreme Destiny Asgard* **[22]**, uma modificação do jogo *With Your Destiny* **[23]** da *JoyImpact* **[24]** pela *Vibrant* **[25]**.

Como se trata de um jogo com mais de 10 anos de vida, existem diversas limitações e portanto o comportamento escolhido para ser alterado foi a movimentação dos NPCs. O processo de avaliação foi feito com desenvolvedores modificando o comportamento escolhido utilizando tanto o processo proposto quanto de seus próprios processos.

1.4 Estrutura do documento

Além deste capítulo de introdução, este trabalho de graduação possui a seguinte estrutura:

Capítulo 2 – O Problema: Este capítulo nos apresenta qual problema será tratado neste trabalho, bem como sua relevância e complexidade no desenvolvimento de jogos. Também explicamos brevemente o que é esperado para sua solução.

Capítulo 3 – Estado da Arte: Este capítulo nos apresenta alguns dos processos já existentes para modificação de jogos MMO sem acesso ao código fonte.

Capítulo 4 – Proposta de Solução: Neste capítulo será explicada a proposta de solução, bem como seu modelo de processo, para realizar uma modificação de comportamento em um *mob*. Também apresentaremos a metodologia utilizada neste trabalho.

Capítulo 5 - Estudo de caso: Neste capítulo faremos um estudo de caso do nosso processo com uma experimentação no jogo Supreme Destiny Asgard.

Capítulo 6 - Resultados: Neste capítulo apresentaremos, e faremos uma breve análise, dos resultados dos nossos testes.

Capítulo 7 – Conclusões: Este capítulo apresenta as contribuições do nosso trabalho, as limitações encontradas na criação do modelo e possíveis trabalhos futuros.

2 O Problema

Ao fazer modificações em jogos de terceiros, assim como nos softwares, os desenvolvedores geralmente não tem acesso ao código fonte, e por tanto precisam utilizar outras formas de acesso ao código para realizar suas modificações. Apesar de ser mais seguro resolver esse problema utilizando apenas ferramentas disponibilizadas pelas empresas desenvolvedoras do jogo, estas ferramentas nem sempre são suficientes para satisfazer as necessidades criativas dos programadores **[16]**, levando a necessidade de um desenvolvimento diretamente no executável. Neste trabalho de graduação, estamos tratando de jogos em que estas ferramentas não foram disponibilizadas, ou que as modificações desejadas estão fora de seu escopo, e portanto outros processos devem ser utilizados.

Apesar de ser uma forte comunidade no desenvolvimento de jogos, a forma mais direta de se tornar um *modder* é sendo autodidata e desenvolvendo práticas estruturadas autorais **[4]**. Além disso, a modificação de comportamento de mobs é um trabalho que exige diferentes técnicas, mecanismos **[4]** e habilidades avançadas de programação **[18]**. A junção de tantas variáveis complexas torna a atividade de modificar comportamentos em mobs sem o código fonte demorada e passiva de um maior risco para inserção de erros no jogo **[11]**. Estes erros afetam diretamente a jogabilidade, diminuindo a imersão dos jogadores e consequentemente sua retenção.

O problema de modificar o comportamento de um NPC sem o código fonte pode ser visto da seguinte forma **[8]**:

1. Definir as posições de memória dos atributos que queremos modificar;
2. Modificar controladamente os valores das posições de memória que encontramos anteriormente;

Utilizando os pontos citados acima, podemos fazer uma adaptação mais específica para entender o problema de modificar a movimentação de um NPC:

1. Definir as posições de memória que armazenam a posição do NPC;
2. Definir uma posição de memória para inserção da nova função de controle de movimento;
3. Criação de nova função update para controle do movimento do NPC;
4. Inserção do novo comportamento do NPC através de alguma DLL do jogo;

Para definir a posição de memória consistente para a inserção da superposição do novo movimento é necessário determinar um local que garanta a

integridade do nosso código, pois escolhendo um local anterior à lógica de movimento nativa do jogo nossa lógica de movimentação seria sobrescrita.

Por existirem diversas maneiras de compilação, o código de baixo nível pode estar diferente em cada situação, gerando a necessidade de uma solução mais genérica. Além disso, por existir uma maior dificuldade para programadores de baixa experiência a solução deve ser de simples entendimento. Este trabalho foca principalmente nas etapas 1, 2 e 4, porém também será feita uma breve explicação da etapa 3.

3 Estado da Arte

O objetivo deste capítulo é apresentar o estado da arte do processo de modificação de jogos MMOs sem acesso ao código fonte, explicando as abordagens propostas e utilizadas em outros artigos.

A pesquisa de Scacchi **[Scacchi et al. 2010]** indica que uma das formas de modificação em jogos mais comuns é através de ferramentas disponibilizadas **[13]** pelas empresas desenvolvedoras dos jogos. Na pesquisa de Hayes **[27]**, são mencionadas algumas dessas ferramentas, como a ferramenta utilizada na série de jogos *Civilization*, que nos possibilita criar novos mapas e desafios para o jogo. Outra ferramenta mencionada é a da série de jogos *Elder Scrolls*, que nos traz mais liberdade na criação de conteúdo, possibilitando desde a criação de novas habilidades nos personagens do jogo quanto a geração de novos tramas para a história.

Sotamaa **[26]** cita em sua pesquisa os chamados *Game Development Kits* (Kits de desenvolvimento para jogos), comuns na modificação de jogos, e cita como exemplo o estúdio Bohemia Interactive **[28]**. O estúdio liberou aos jogadores parte de suas ferramentas de desenvolvimento do jogo *Operation Flashpoint Resistance*, como Kit de desenvolvimento, para facilitar a criação dos mods e fortalecer a comunidade do jogo. Essas ferramentas, apesar de trazerem mais segurança na modificação, pois são feitas exclusivamente e diretamente para os jogos em questão, trazem uma limitação do que os modders podem fazer.

Nick Cano, por sua vez, nos apresenta no capítulo 1 do seu livro “Game Hacking: Developing Autonomous Bots for Online Games” **[8],** um processo chamado de *Memory Scanner* (Escaneamento de Memória) utilizando uma ferramenta mais genérica para monitorar posições de memória. Ele explica como a ferramenta *Cheat Engine* **[19]** pode ser usada para encontrar valores esperados na memória, buscando por filtragens como "valores maiores que" ou "valores menores que". Além disso, ele apresenta diversas outras funcionalidades dessa ferramenta que podem ser utilizadas dependendo da forma como você deseja escanear a memória e as informações que você já possui. O processo, apesar de bastante útil e robusto para encontrar os endereços de memória que guardam a posição atual e final de NPC, não é suficiente para encontrar a posição de memória que necessitamos para a inserção do novo comportamento.

Ainda no mesmo livro **[8]**, somos apresentados outra ferramenta de grande utilidade para executar ações de engenharia reversa, o *OllyDbg* **[20]**. Ele explica diversas de suas principais funcionalidades e como elas poderiam ser úteis para

análise de códigos de baixo nível. Posteriormente ele nos mostra uma forma de como encontrar e modificar a quantidade de vida de um personagem utilizando tal ferramenta. Ele realiza uma busca pela *string* "*Health*", e analisa o código encontrado buscando entender como o mesmo seria escrito em mais alto nível para facilitar seu entendimento. Por fim ele conclui a partir de sua análise os endereços que guardam os valores de "vida atual" e "vida máxima" do personagem, possibilitando mudanças posteriores. A busca neste processo se assemelha com parte da proposta deste artigo, porém a busca por *strings*, como citada pelo autor, nem sempre é suficiente, e outras alternativas devem ser buscadas.

4 Proposta de Solução

4.1 Metodologia

4.1.1 BPMN

Para facilitar o entendimento do nosso processo escolhemos modelá-lo em uma forma de representação gráfica. Modelamos as tarefas envolvidas utilizando BPMN (*Business Process Model Notation*) versão 2.0 **[6]**, pois é um modelo de fácil diagramação e entendimento, e conciso na sua representação **[7]**. Para desenvolvê-lo utilizamos uma ferramenta *online* e gratuita **[9]** e seus principais símbolos podem ser vistos na **Tabela 1 [10]** abaixo.

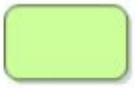
BPMN element	Description
 Start Event Interm. Event End Event	The Start Event indicates where a particular process will start. Intermediate Events occur between a Start Event and an End Event and will affect the flow of the process. The End Event indicates where a process will end.
 Task	A task is an atomic activity within the process. It represents work to be performed. A task will be activated when one of its incoming sequence flows is triggered and will trigger all its outgoing sequence flows when it is finished.
 AND Gateway	An AND Gateway will trigger all outgoing flows, when all incoming flows are triggered.
 XOR Gateway	An XOR Gateway will trigger one of its outgoing flows when one of its incoming flows is triggered.
 Subprocess	Subprocesses are used to support hierarchy. A Subprocess contains its own Start and End Event.
 Sequence Flow	The Sequence Flow shows the order in which activities will be performed in a Process.

Tabela 1 - Elementos BPMN **[10].**

4.1.2 Modelagem

O processo metodológico escolhido para este trabalho de graduação pode ser definido nas seguintes etapas **(Figura 3)**:

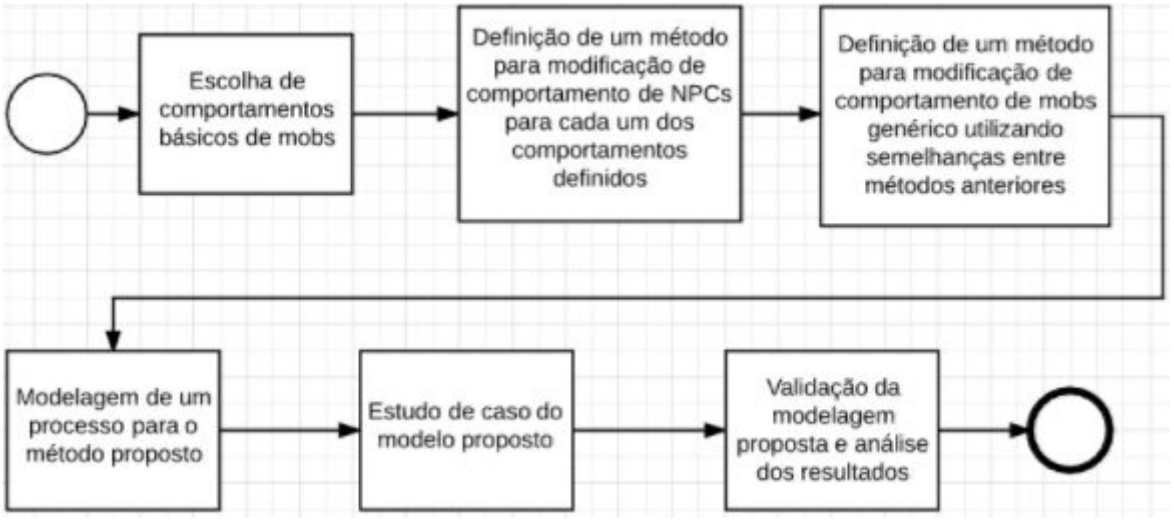


Figura 3 - Etapas do processo metodológico escolhido para este trabalho de graduação.

Os comportamentos básicos que escolhemos foram os comportamentos mais comuns em mobs de *quests* e *dungeons*, eles são:

- *Patrol*: Realiza um caminho cíclico;
- *Follow*: Segue algum *mob* ou jogador;
- *Guard*: Protege algum tesouro ou objeto;
- *Guide*: Guia o jogador por um caminho, parando caso o jogador se distancie.

Semelhantemente à metodologia Ronimo **[15]** para criação de novas features, discutimos com especialistas uma sequência de tarefas, de definição mais explícita, que seriam necessárias para modificação de cada um destes comportamentos de mobs. Dessa forma, montamos um processo individual e mais específico para cada um no contexto que estamos inseridos.

Para robustez do nosso método, adaptamos também outros processos presentes na literatura **[8]**, porém buscando torná-las mais direcionadas para a mudança proposta. Analisamos cada processo definido anteriormente, como um *overlap*, focando em suas semelhanças para gerar um novo processo mais amplo para comportamentos de mobs. Nos passos de maior diferença entre comportamentos, buscamos criar tarefas genéricas, diminuindo a especialização, mas facilitando posterior escalabilidade.

O jogo selecionado para experimentação,o *Supreme Destiny Asgard*, foi escolhido por: se qualificar em todos os requisitos necessários para este experimento; ter sua relevância no cenário nacional com milhares de jogadores ativos diariamente; suas limitações comportamentais dadas as limitações da época que foi desenvolvido; e sua facilidade de acesso aos desenvolvedores e especialistas envolvidos nos experimentos e validação. A ferramenta de análise de executável utilizada foi o *OllyDBG* **[20]**, pela experiência de uso e representatividade na literatura **[8].**

4.2 Solução Proposta

Para geração da nossa solução definimos três grandes atividades atreladas na mesma **(Figura 4)**: Encontrar a construção do pacote de movimento; Encontrar as posições de memória necessárias; Criar a função de *hook* na DLL.

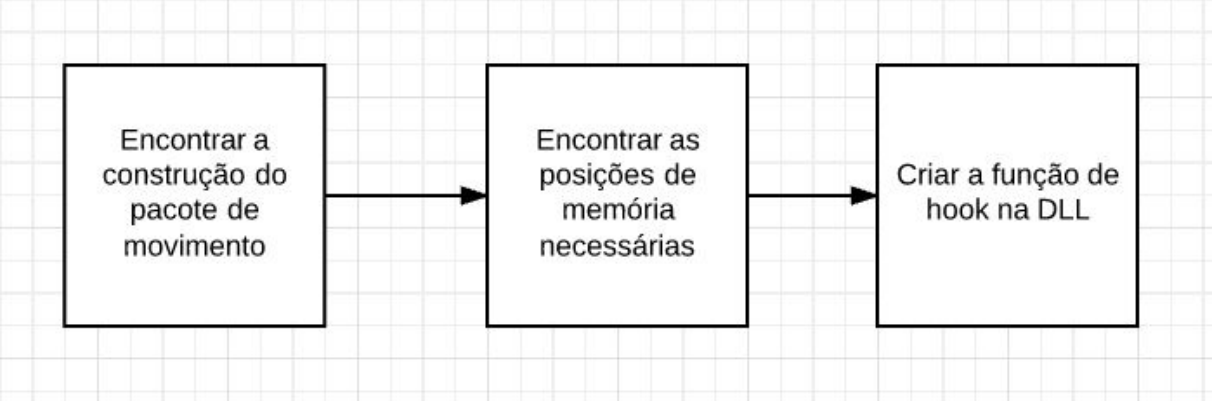


Figura 4 - Visão geral das subseções do modelo proposto.

A partir das atividades mostradas na **Figura 4** escolhemos comportamentos comuns em mobs de *quest* e definimos uma sequência de passos para modificar e implementar cada um destes, como pode ser visto abaixo:

Comportamento *Patrol*:

- Buscar pelo Id do pacote de movimento;
- Colocar *breakpoint* em uma das opções encontradas;
- Forçar servidor a enviar um pacote de movimento;

Parou no *breakpoint*? Sim?

- Modificar controladamente a posição do mob;
- Buscar nas instruções de acesso à memória o valor esperado para a nova e a antiga posição do mob;
- Analisar o cálculo do endereço base;

- Definir um endereço de memória para inserção de um novo movimento;
- Criar uma função de comportamento *Patrol* na dll que selecione e controle posições pré-definidas cíclicas;
- Criar um *hook* na DLL no endereço proposto, sobrescrevendo os valores nas memórias encontradas pelos valores da função de comportamento Patrol

Comportamento *Follow*:

- Buscar pelo Id do pacote de movimento;
- Colocar *breakpoint* em uma das opções encontradas;
- Forçar servidor a enviar um pacote de movimento;

Parou no *breakpoint*? Sim?

- Modificar controladamente a posição do mob;
- Buscar nas instruções de acesso à memória o valor esperado para a nova e a antiga posição do mob;
- Analisar o cálculo do endereço base;
- Definir um endereço de memória para inserção de um novo movimento:
- Utilizar o endereço base para encontrar a posição de outro mob/player;
- Criar uma função de comportamento *Follow* na dll que gere valores de posição a partir da posição de outro *mob* ou jogador
- Criar um *hook* na DLL no endereço proposto, sobrescrevendo os valores nas memórias encontradas pelos valores da função de comportamento *Follow*

Comportamento *Guard*:

- Buscar pelo Id do pacote de ataque:
- Colocar *breakpoint* em uma das opções:
- Forçar servidor a enviar um pacote de ataque;

Parou no breakpoint? Sim?

- Modificar controladamente o ataque do mob;
- Buscar nas instruções de acesso à memória o valor esperado para o identificador do jogador atacado;
- Definir um endereço de memória para inserção de um novo comportamento;
- Criar uma função de comportamento *Guard* na dll que altere (ou insira) o valor do identificador do jogador atacado por algum definido pela interação com uma armadilha (ou recompensa)

- Criar um *hook* na DLL no endereço proposto, sobrescrevendo os valores nas memórias encontradas pelos valores da função de comportamento Guard

Comportamento *Guide*:

- Buscar pelo Id do pacote de movimento:
- Colocar *breakpoint* em uma das opções:
- Forçar servidor a enviar um pacote de movimento;

Parou no breakpoint? Sim?

- Modificar controladamente a posição do mob;
- Buscar nas instruções de acesso à memória o valor esperado para a nova e a antiga posição do *mob*;
- Analisar o cálculo do endereço base;
- Definir um endereço de memória para inserção de um novo movimento:
- Utilizar o endereço base para encontrar a posição de outro *mob* ou jogador;
- Criar uma função de comportamento *Guide* na DLL que gere valores de posição a partir da posição de outro *mob* ou jogador;
- Criar um *hook* na DLL no endereço proposto, sobrescrevendo os valores nas memórias encontradas pelos valores da função de comportamento Guide

Como queríamos um modelo mais genérico, buscamos os pontos de incidência desses processos, substituindo os que mais se diferenciavam por passos genéricos que pudessem ser escalados posteriormente:

Processo proposto:

- Buscar pelo Id do pacote de movimento;
- Colocar *breakpoint* em uma das opções;
- Forçar servidor a enviar um pacote de movimento;

Parou no breakpoint? Sim?

- Modificar controladamente a posição do mob;
- Buscar nas instruções de acesso à memória o valor esperado para a nova e a antiga posição do *mob*;
- Definir o id dado ao mob no pacote de movimento;
- Analisar o cálculo do endereço base;
- Definir um endereço de memória para inserção de um novo comportamento;
- Criar uma função de comportamento *Move* na dll que selecione e controle posições pré-definidas;

- Criar um *hook* na DLL no endereço proposto, sobrescrevendo os valores nas memórias encontradas pelos valores da função de comportamento Move;

A solução proposta consiste de um modelo de processo, como um passo a passo, construído para facilitar o entendimento e acelerar o processo da modificação do comportamento de um NPC em um jogo MMO, que utiliza DLLs para inserção de código e tendo o id dos seus pacotes.

O comportamento escolhido como foco desta solução foi “movimento” e o modelo de processo criado pode ser visto abaixo **(Figura 5)**:

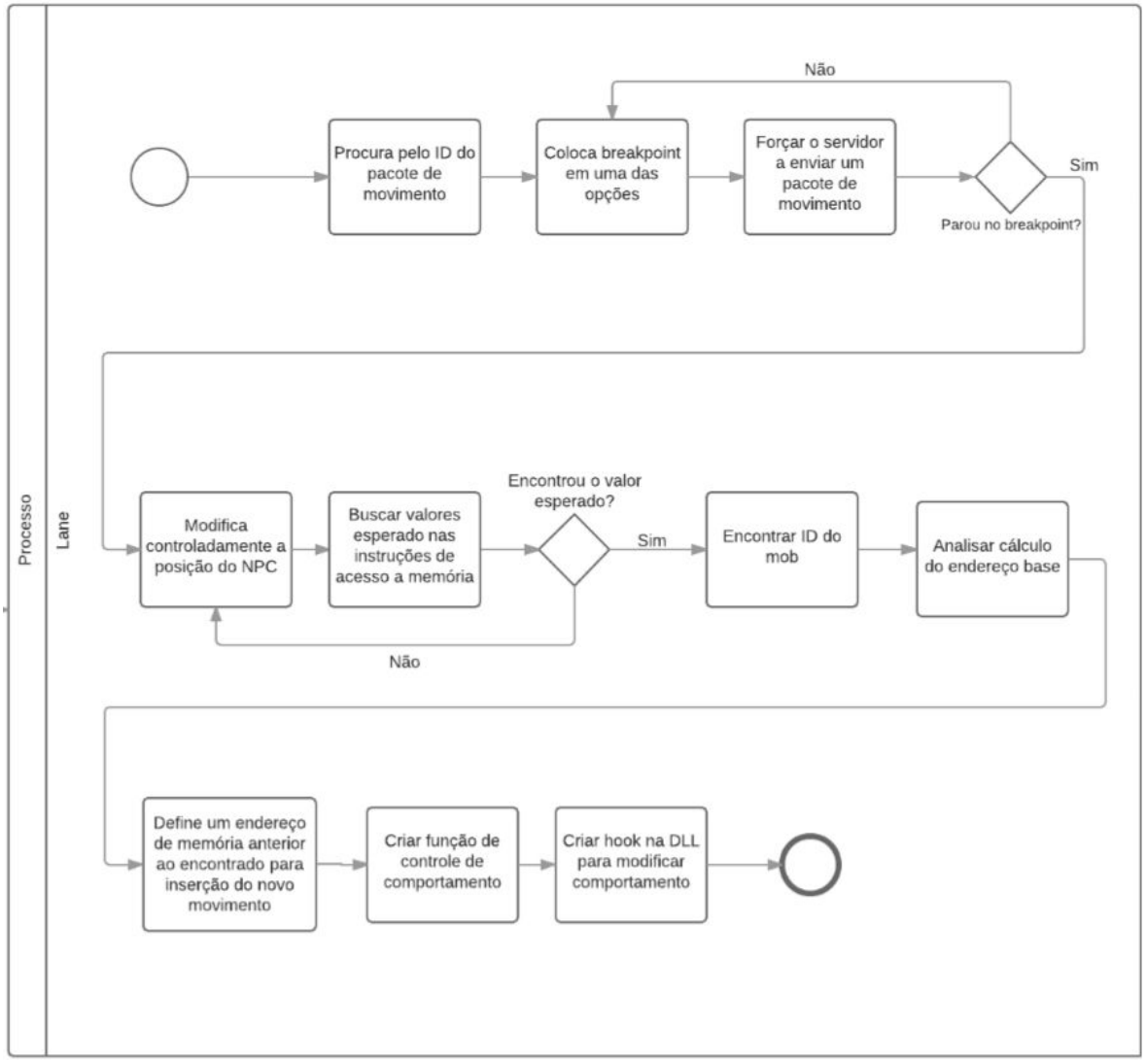


Figura 5 - O processo proposto em notação BPMN 2.0

Para realizar estas tarefas adequadamente você necessitará de algum programa ou ferramenta que possibilite buscar constantes e colocar *breakpoints*. Abaixo pode ser

encontrada uma breve explicação de cada tarefa definida neste modelo e a subseção em que está inserida:

Encontrar a construção do pacote de movimento:

- Busca pelo Id do pacote de movimento:

Utilizando alguma ferramenta, de escolha própria, para análise de código do executável você deve realizar uma busca pelo identificador do pacote de movimento no código do servidor. Os resultados esperados desta busca deverão ser posições de memória com instruções que estão utilizando o identificador.

- Colocar *breakpoint* em uma das opções:

A função desta tarefa é definir qual das instruções encontradas no evento anterior faz parte da montagem do pacote de movimento. Para realizar tal tarefa devemos colocar um *breakpoint* em uma (ou mais) das instruções e checar qual destes *breakpoints* será acionado quando o pacote de movimento for enviado pelo servidor.

- Enviar um pacote de movimento:

A função desta tarefa é garantir o envio do pacote de movimento, do servidor ao cliente, para que o passo anterior funcione corretamente. Para realizar este passo é preferível um ambiente controlado, e com apenas um *mob* ou NPC que se movimenta, e aguardar sua movimentação, ou caso possua a possibilidade, forçar o mob a se mover.

Encontrar as posições de memória necessárias:

- Modificar controladamente a posição do mob:

A função deste passo é garantir um valor esperado para a posição final (e/ou inicial) do mob e possibilitar uma análise mais robusta das instruções. Para realizar este passo precisamos garantir os valores de posição na movimentação de um *mob*, podendo este ser criado para o teste ou nativo do jogo.

- Buscar nas instruções de acesso à memória o valor esperado para a nova e a antiga posição do *mob*;

A função deste passo é definir qual das instruções estão acessando os valores de posição, da estrutura do NPC, que garantimos no passo anterior. Para facilitar este passo focamos em instruções de acesso a memória que utilizem constantes esdrúxulas, pois é esperado que o valor da posição do NPC estará no mesmo *offset* para todas as suas estruturas, então bastaria ao compilador

determinar a posição inicial e usar a mesma constante (que é o valor do *offset*) para sempre acessá-los.

- Encontrar o id dado ao mob no pacote de movimento;

A função deste passo é definir qual o id dado ao mob no pacote de movimento, e se este id é semelhante ao esperado ou é gerado em tempo de execução. Esta informação poderá ser útil para o entendimento do cálculo de endereço base.

- Analisar o cálculo do endereço base;

A função deste passo é analisar e entender como é feita a indexação, e portanto como é definido o endereço base do mob. Este cálculo é de suma importância para as funções que construiremos posteriormente.

- Definir um endereço de memória para inserção de um novo movimento:

A função deste passo é definir uma posição de memória que seja próxima o suficiente da criação do pacote de movimento para evitar que os novos valores sejam sobrescritos. Devemos levar em consideração que este endereço deve, no entanto, estar antes das instruções que copiam da memória os valores de posição finais (ou atuais) da movimentação.

Criar a função de *hook* na DLL:

- Criar função de controle de comportamento na dll:

A função deste passo é criar uma nova função *update* para o mob que controlará e definirá as suas novas posições a partir de um comportamento pré definido.

- Criar hook na dll para inserir novo comportamento:

A função deste passo é criar um método em alguma das DLLs do jogo que utilize a função de controle, e as informações obtidas nos passos anteriores para sobrescrever os valores presentes nos endereços de memórias de posição do *mob*. Para realizar este passo precisamos dos endereço de posição atual (e possivelmente final) do mob, bem como seu id e o cálculo de endereçamento base.

5 Estudo de Caso

5.1 *Supreme Destiny Asgard* (WYD *Mod*)

Para a experimentação e avaliação da solução proposta neste trabalho de graduação utilizamos o jogo *Supreme Destiny Asgard* **(Figura 6)**, um *mod* do jogo With Your Destiny (WYD) **(Figura 7)**. Este jogo, publicado em 2003, possui diversas limitações, e o movimento de seus mobs se limita a rotas estáticas, e comportamentos agressivo ou pacífico. O mod que utilizaremos, de direitos da *Vibrant*, possui desenvolvedores e servidores localizados no brasil com operação focada em eventos semanais. Sua operação de desenvolvimento, é feita quase que em totalidade no executável por não possuírem acesso ao código fonte. Pelas limitações que possui e pelas suas características este mod foi um perfeito candidato para nosso estudo de caso.

Figura 6 - *Screenshot* do jogo *Supreme Destiny Asgard (Mod* do WYD). *(Fonte: Google Imagens)*

Figura 7 - *Screenshot* do jogo *With Your Destiny. (Fonte: Google Imagens)*

5.2 Experimentação

Para a experimentação do nosso modelo proposto utilizamos um ambiente controlado no jogo *Supreme Destiny Asgard*, utilizando permissões de GM e nos limitando a um mapa com apenas um mob para diminuir a quantidade de pacotes de ações indesejadas que receberíamos durante o processo. Para nosso teste focamos em um mob específico criado pelas ferramentas nativas do jogo, o qual sabíamos as informações de comportamento.

Inicialmente utilizamos um processo semelhante ao aplicado por Nick Cano **[8]** no programa OllyDbg, mas diferente de procurar por uma *string*, fizemos uma busca por uma constante, pois o id do nosso pacote de movimento é um valor inteiro. Anexamos o programa ao servidor local do jogo e fizemos uma busca por todas as constantes *0x36C **(Figura 8)***, encontrando algumas opções como mostrado na **Figura 9** abaixo:

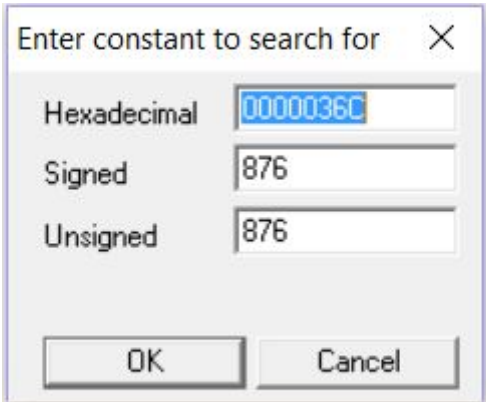


Figura 8 – Janela de busca por constantes no OllyDbg.

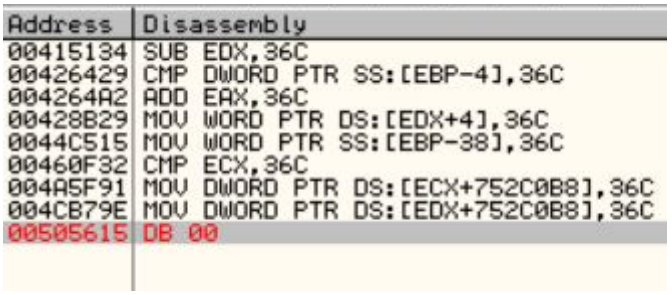


Figura 9 – Imagem parcial da janela de resultados na busca por constantes no OllyDbg.

Qualquer um desses endereços de memória mostrados na coluna “*Address*” vista na **Figura 9** poderiam ser corretos, portanto colocamos um *breakpoint* em um deles e aguardamos nosso mob de teste se mover, gerando o envio de um pacote de movimento pelo servidor e checando se o *breakpoint* seria acionado. Após algumas iterações o programa parou no *breakpoint (***Figura 10***)* e encontramos a única destas instruções que é referenciada antes de todo envio de um pacote de movimento. Concluímos então que este é o ponto de contrução do pacote, e a partir deste ponto podemos descobrir qual o endereço possui os valores que iriamos modificar.

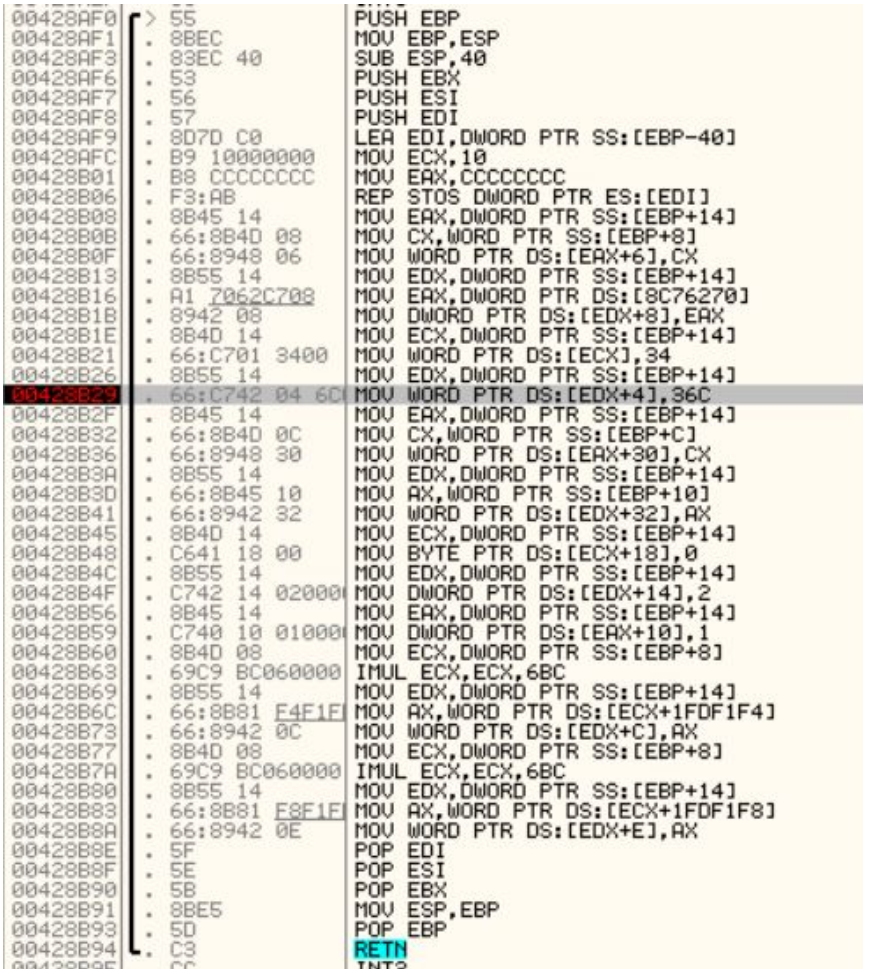


Figura 10 – Janela de endereços virtuais no OllyDBG, programa paralisado no *breakpoint.*

Observando o código em questão **(Figura 10)**, concluímos que a construção do pacote estava dentro de uma função, e sendo esta uma função para criação de um pacote de movimento, os valores da posição final do mob deviam ser obtidos dentro desta função ou pelos parâmetros da mesma. Como necessitaríamos uma posição de memória anterior a cópia dos valores de posição do mob preferimos iniciar nossa checagem a partir dos parâmetros dessa função e assim colocamos um novo *breakpoint* no seu início para encontrar seu endereço de retorno e consequentemente onde é chamada.

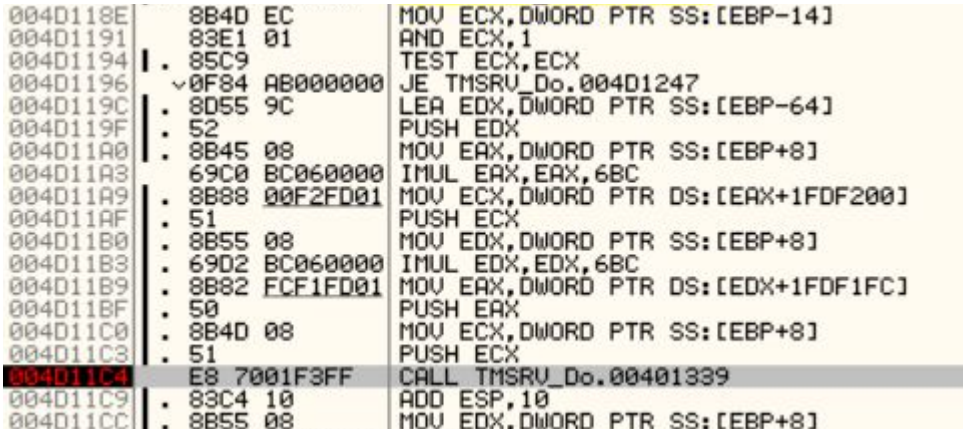


Figura 11 – Janela de endereços virtuais no OllyDbg, programa paralisado no na chamada da função.

A partir do endereço de retorno analisamos as instruções anteriores, focando nas de acesso à memória que utilizavam constantes mais esdrúxulas (que seriam os possíveis *offsets* no acesso à estrutura do mob). Como pode ser visto na **Figura 11** algumas destas instruções utilizavam constantes semelhantes às encontradas anteriormente dentro da função de construção do pacote (*0x1FDF1F4; 0x1FDF1F8; 0x1FDF1FC* e *0x1FDF200*).

Aguardamos o NPC se movimentar para uma posição do mapa com valores esperados e utilizando *breakpoints* procuramos nestas instruções estes valores no acesso à memória. Estes valores (neste caso eram esperados valores entre 0x840 a 0x842) podem ser observados na primeira linha do painel de dicas do OllyDbg como visto na **Figura 12** abaixo:

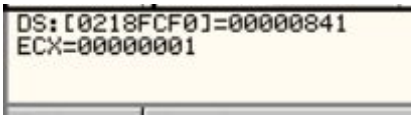


Figura 12 – Painel de dicas no OllyDbg.

Encontrando estes valores, identificamos as instruções de acesso à memória do mob, e quais os valores são utilizados como offset para acesso às suas posições finais. A partir da semelhança entre as constantes citadas anteriormente, presentes antes da chamada da função e dentro da mesma, realizamos mais testes e identificamos que os valores internos representavam os *offsets* das posições atuais do mob.

Outra semelhança nos chamou atenção, e após algumas iterações checando os valores na instrução que utiliza a constante 0x6BC, tanto antes da chamada da função quanto dentro, descobrimos que ela utiliza o id do mob para realizar o cálculo

do endereço base de sua indexação, que posteriormente é utilizado no acesso à posição do mob.

Sabendo a primeira instrução de busca pelo valor da posição do mob na memória pudemos escolher de forma segura uma posição de memória anterior a esta que seria o ponto de inserção para o nosso novo comportamento. A posição de memória escolhida foi *0x04D119C*, e utilizamos a função de inserção de código (*JMP_NEAR*) já existente na DLL **(Figura 13)** para sua inserção no executável.

```
JMP_NEAR(0x04D119C, NKD_MonsterMovement, 2);
```

Figura 13 - Função de *hook* da DLL, para inserção de código na posição encontrada.

Criamos então uma função **(Figura 14)** que utiliza valores predefinidos para determinar o comportamento do mob. Para esta experimentação definimos um comportamento semelhante ao *Patrol* citado anteriormente neste trabalho.

Utilizando o código do cálculo de endereçamento base encontrado anteriormente, assim como o das instruções para acesso a posição do mob criamos na DLL uma função que utiliza a nossa função de comportamento para sobrescrever a movimentação do mob **(Figura 15)**.

```
void HKD_MonsterMovement_Direction()
{
    static int varDecide = 0;

    if(varDecide == 0)
    {
        g_mobToX = 4;
        g_mobToY = 0;
        varDecide = 1;
    }
    else
    {
        g_mobToX = -4;
        g_mobToY = 0;
        varDecide = 0;
    }
}
```

Figura 14- Função de comportamento criada na DLL.

```
__declspec(naked) void NKD_MonsterMovement()
{
    __asm
    {
        CALL HKD_MonsterMovement_Direction
        MOV EDX,DWORD PTR SS:[EBP+8]
        IMUL EDX,EDX,0x6BC
        MOV EAX,DWORD PTR DS:[EDX+0x1FDF1F4]
        ADD EAX, g_mobToX
        MOV DWORD PTR DS:[EDX+0x1FDF1FC],EAX

        MOV EDX,DWORD PTR SS:[EBP+8]
        IMUL EDX,EDX,0x6BC
        MOV EAX,DWORD PTR DS:[EDX+0x1FDF1F8]
        ADD EAX, g_mobToY
        MOV DWORD PTR DS:[EDX+0x1FDF200],EAX

        LEA EDX,DWORD PTR SS:[EBP-0x64]
        PUSH EDX
        MOV EAX,DWORD PTR SS:[EBP+8]
        PUSH 0x04D11A3
        RETN
    }
}
```

Figura 15 - Função de modificação de valores criada na DLL.

6 Resultados

6.1 Metodologia

Para avaliação do nosso modelo de processo realizamos um teste A/B com programadores buscando descobrir se utilizando nosso processo o desenvolvimento da solução e o entendimento do problema seria mais rápido que utilizando seus próprios processos. Os programadores selecionados para participar no nosso teste tinham um conhecimento prévio em assembly, e foram requisitados a realizar o processo de modificação de comportamento de um NPC no jogo Asgard.

Em um teste A/B clássico cada programador deveria realizar o próprio *ad hoc* e em seguida realizar o processo proposto, porém, dado que o conhecimento obtido no primeiro processo realizado iria facilitar o segundo, tivemos que adaptar o teste A/B. Portanto, para nosso teste A/B selecionamos 3 dos programadores para testar nosso processo e requisitamos que os outros 2 realizassem um próprio *ad hoc*.

Registramos desses programadores o tempo de desenvolvimento e a quantidade de erros gerados. Definimos como erro a quantidade de vezes que o programador passou para o próximo passo com valores ou posições de memória incorreta.

6.2 Resultados e análise

Abaixo pode ser vista uma tabela **(Tabela 2)** com o tempo estimado pelos programadores para o desenvolvimento, e o tempo real que levaram para o desenvolvimento com próprio *ad hoc*:

	Tempo estimado com próprio *ad hoc* (em horas)	Tempo real usando proprio *ad hoc* (em horas)
Desenvolvedor 1	6h	10h
Desenvolvedor 2	10h	9h

Tabela 2 - Tempo estimado com próprio *ad hoc* e tempo real de desenvolvimento.

Como citado anteriormente, requisitamos a outros programadores que estimassem um tempo para a atividade em questão (modificar o comportamento de um NPC) utilizando seus próprios *ad hoc*. Após essa estimativa, apresentamos nosso modelo de processo e pedimos que estimassem a mesma atividade com o modelo proposto. Pedimos então que eles utilizassem o processo proposto, registrando as dificuldades encontradas, o tempo levado em cada etapa, e o tempo total para seu desenvolvimento.

Para esta etapa foram coletadas respostas referentes a 3 programadores, sendo 2 destes com menos de 1 ano de experiência na área, os dados podem ser vistos na **Tabela 3** abaixo:

	Tempo estimado com próprio *ad hoc* (em horas)	Tempo estimado com processo proposto (em horas)	Tempo real com processo proposto (em horas)
Desenvolvedor 3	12h	5h	6h
Desenvolvedor 4	10h	10h	3.5h
Desenvolvedor 5	8h	2h	4h

Tabela 3 - Tempo estimado com e sem o processo proposto, e tempo real de desenvolvimento com processo proposto.

Na **Tabela 4** registramos o tempo de desenvolvimento para cada subseção do modelo proposto: Encontrar a construção do pacote de movimento; Encontrar as posições de memória necessárias; Criar a função de *hook* na DLL:

	Tempo na parte 1	Tempo na parte 2	Tempo na parte 3
Desenvolvedor 3	2h	3h	1h
Desenvolvedor 4	0.5h	2h	1h
Desenvolvedor 5	1h	2h	1h

Tabela 4 - Tempo de desenvolvimento em cada subseção do modelo proposto.

A **Tabela 5** mostra a quantidade de erros gerados durante o desenvolvimento em cada subseção citada anteriormente:

	Erros na parte 1	Erros na parte 2	Erros na parte 3
Desenvolvedor 3	3	4	1
Desenvolvedor 4	0	0	0
Desenvolvedor 5	1	2	0

Tabela 5 - Quantidade de erros em cada subseção do modelo proposto.

A **Tabela 4** nos indica que a parte mais demorada do processo está na segunda subseção, o que é ratificado pela quantidade de erros apresentados na mesma subseção da **Tabela 5**. Estes resultados foram exatamente como esperados, devido a dificuldade de gerar tarefas tanto genéricas quanto diretas nesta subseção.

A partir dos resultados da **Tabela 3** podemos ver que, ao seguir o método proposto a estimativa de tempo diminui drasticamente para programadores com menor experiência na área. Este resultado, apesar de inconclusivo, aponta que o modelo proposto facilita o entendimento do processo de modificação de comportamento, sendo ainda mais impactante em programadores iniciantes. Uma comparação entre os resultados obtidos utilizando próprio *ad hoc* e utilizando nosso processo pode ser visto na **Tabela 6** abaixo, nos apontando que a facilidade de entendimento, como citada anteriormente, influencia diretamente na redução do tempo de desenvolvimento.

Tempo médio de desenvolvimento utilizando próprio *ad hoc*	Tempo médio de desenvolvimento utilizando processo proposto
9.5 horas	4.5 horas

Tabela 6 - Comparação do tempo médio utilizando próprio *ad hoc* e utilizando processo proposto.

7 Conclusões

7.1 Contribuições

O processo de modificação em um executável é uma atividade que exige criatividade e conhecimento, e portanto é bastante afetada pela experiência do desenvolvedor. O modelo proposto, diferente do que está presente na literatura, não foca em explicar como funcionam ferramentas para a engenharia reversa do processo, mas sim criar um processo genérico que utilizando de engenharia reversa facilita o entendimento das tarefas que devem ser executadas para modificações em personagens de jogos online, acelerando o processo de desenvolvimento e diminuindo os custos de produção. Devido a nossa metodologia de modelagem o modelo é genérico, e com algumas adaptações pode ser usado para outros tipos de modificações, não se limitando exclusivamente à comportamento de NPCs.

Além disso, o modelo pode ser adaptado para especializações dos mais diversos jogos e tipos de compilações, diminuindo a quantidade de erros gerados e acelerando o aprendizado de novos programadores na área, como um guia para mods de comportamento em jogos MMO.

7.2 Limitações

Os principais desafios, que encontramos, para criar um modelo de processo genérico para modificação de comportamento de NPCs em qualquer jogo foram dois: Selecionar as tarefas mais apropriadas para o modelo e avaliar a efetividade do modelo proposto.

Selecionar as tarefas mais apropriadas para o modelo é uma atividade muito complexa. As tarefas precisam ser diretas o suficiente para tornar seu entendimento fácil, mas ao mesmo tempo precisam ser genéricas o suficiente para que funcionem para todos, ou a maioria, dos jogos. Cada tarefa precisa ser balanceada entre explicar exatamente o que precisa ser feito nesse passo e o quanto sua explicação confundiria o desenvolvedor (dependendo do método de compilação que foi utilizado para gerar o executável). Por exemplo, a indexação das estruturas de mob na memória podem ser feitas das mais diferentes maneiras, e portanto não podemos ser muito específicos na tarefa de analisar sua indexação pois podemos confundir e guiar de maneira errada o desenvolvedor, porém explicando com mais detalhes, a

atividade se tornaria muito mais simples, consequentemente seu desenvolvimento seria ainda mais rápido e menos erros seriam gerados.

O outro desafio na modelagem do processo de modificação de comportamento de NPCs é escolher a forma correta de avaliar a efetividade do modelo. Devido a dificuldade no aprendizado da engenharia reversa, e a falta de modelos como o proposto neste trabalho, os *modders* utilizam muito mais ferramentas disponibilizadas pelas empresas do que modificações diretamente no executável e por ser uma atividade diretamente relativa a experiência na área, não temos como controlar o efeito disso na utilização do modelo.

7.3 Trabalhos Futuros

Existem diversos pontos no trabalho que podem sofrer melhorias. A princípio, testar nosso modelo em diversos outros jogos, para posteriormente, após análise dos novos dados, adaptá-lo de forma que identifique formas de compilação diferente e facilite o processo a partir disso.

A escolha das tarefas da segunda subseção (da nossa solução) é uma etapa que ainda precisa ser melhor estruturada, porém ainda não há uma definição clara de como estruturá-las adequadamente. Outra melhoria muito importante seria evitar colisões durante nosso comportamento de movimentação, gerando movimentos ainda mais robustos e complexos.

Referências Bibliográficas

[1] SCACCHI, Walt. Computer game mods, modders, modding, and the mod scene. **First Monday**, v. 15, n. 5, 2010.

[2] TAYLOR, T. L. The assemblage of play. **Games and Culture**, v. 4, n. 4, p. 331-339, 2009.

[3] SOTAMAA, Olli. When the game is not enough: Motivations and practices among computer game modding culture. **Games and Culture**, v. 5, n. 3, p. 239-255, 2010.

[4] SCACCHI, Walt. Modding as a basis for developing game systems. In: **Proceedings of the 1st international workshop on Games and software engineering**. ACM, 2011. p. 5-8.

[5] SCACCHI, Walt. Modding as an open source approach to extending computer game systems. **Open Source Systems: Grounding Research**, p. 62-74, 2011.

[6] O. M. G. Omg, R. Parida, and S. Mahapatra. Business Process Model and Notation (BPMN) Version 2.0. Technical Report January, 2011.

[7] TANG, Stephen et al. Towards a domain specific modelling language for serious game design. In: **6th International Game Design and Technology Workshop, Liverpool, UK**. 2008.

[8] CANO, Nick. **Game Hacking: Developing Autonomous Bots for Online Games**. No Starch Press, 2016.

[9] *Lucid Chart. Disponível em: <https://www.lucidchart.com>. Acesso em: 6 jul. 2017.*

[10] RAEDTS, Ivo et al. Transformation of BPMN Models for Behaviour Analysis. **MSVVEIS**, v. 2007, p. 126-137, 2007.

[11] BERDAJS, J.; BOSNIĆ, Z. Extending applications using an advanced approach to dll injection and api hooking. **Software: Practice and Experience**, v. 40, n. 7, p. 567-584, 2010.

[12] SANTOS, André Pequeno dos; AZEVEDO, José. Fan culture e mods: a dimensão do multiverso cultural dos games na internet. **Videojogos 2016: atas**, 2016.()

[13] HACKMAN, Eleonora; BJÖRKQVIST, Ulfrik. Modders of Skyrim: Motivations and Modifications: A qualitative study of what motivations and modifications the modders of Elder Scrolls: Skyrim exhibit. 2014.

[14] LAUKKANEN, Tero. Modding Scenes-Introduction to user-created content in computer gaming. 2005.

[15] The Ronimo Coding Methodology. Disponível em: <http://www.gamasutra.com/blogs/JoostVanDongen/20170703/300968/The_Ronimo_coding_methodology.php>. Acesso em: 4 jul. 2017.

[16] Berdajs, J., and Z. Bosnić. "Extending applications using an advanced approach to dll injection and api hooking." *Software: Practice and Experience* 40.7 (2010): 567-584.

[17] McKenzie, Pamela J., et al. "User-generated online content 1: Overview, current state and context." *First Monday* 17.6 (2012).

[18] MONTERRAT, Baptiste; LAVOUÉ, Elise; GEORGE, Sébastien. Learning Game 2.0: support for game modding as a learning activity. In: **Proceedings of the 6th European Conference on Games Based Learning**. 2012. p. 340-347.

[19] Cheat Engine. Disponível em: <http://www.cheatengine.org/>. Acesso em: 6 jul. 2017.

[20] OllyDbg. Disponível em: <http://www.ollydbg.de/>. Acesso em: 6 jul. 2017.

[21] Reverse Engineering with OllyDbg. Disponível em: <https://erichokanson.me/2015/04/17/reverse-engineering-with-ollydbg//>. Acesso em: 7 jul. 2017.

[22] Supreme Destiny Asgard. Disponível em: <http://asgard.vibrant3g.com/wyd/pt/>. Acesso em: 9 jul. 2017.

[23] With Your Destiny. Disponível em: <https://pt.wikipedia.org/wiki/With_Your_Destiny>. Acesso em: 9 jul. 2017.

[24] Joy Impact Co., Ltd. Disponível em: <http://www.mobygames.com/company/joyimpact-co-ltd>.

Acesso em: 9 jul. 2017.
[25] Vibrant Games. Disponível em: <https://games.vibrant3g.com/>. Acesso em: 9 jul. 2017.
[26] SOTAMAA, Olli. Playing it my way? Mapping the modder agency. In: **Internet Research Conference**. 2004. p. 19-22.9.
[27] HAYES, Elisabeth. Game content creation and it proficiency: An exploratory study. **Computers & Education**, v. 51, n. 1, p. 97-108, 2008.
[28] Bohemia Interactive. Disponível em: <https://www.bistudio.com/>. Acesso em: 9 jul. 2017.