
Universidade Federal de Pernambuco

Centro de Informática

Graduação em Ciência da Computação

# Ferramenta de Pré-Análise para Auxiliar o Designer no balanceamento de Jogos

AUTOR: ERIC ÂNDERSEN FREITAS E SILVA

ORIENTADOR: DR. GEBER LISBOA RAMALHO

CO-ORIENTADOR: DR. MARCO TÚLIO CARACIOLO FERREIRA ALBUQUERQUE

TRABALHO DE GRADUAÇÃO

Recife, Julho de 2017

Eric Ândersen Freitas e Silva

# Ferramenta de Pré-Análise para Auxiliar o Designer no balanceamento de Jogos

*Trabalho de Conclusão de Curso apresentado à Universidade Federal de Pernambuco – UFPE, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.*

Universidade Federal de Pernambuco
Centro de Informática

Orientador: Dr. Geber Lisboa Ramalho

Co-orientador: Dr. Marco Túlio Caraciolo Ferreira Albuquerque

Recife – Pernambuco
Julho de 2017

Eric Ândersen Freitas e Silva

# Ferramenta de Pré-Análise para Auxiliar o Designer no balanceamento de Jogos

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

**BANCA EXAMINADORA:**

Orientador: _____
Dr. Geber Lisboa Ramalho (UFPE-CIn)

Avaliadora: _____
Dra. Patricia Cabral de Azevedo Restelli Tedesco (UFPE-CIn)

Recife, 11 de Julho de 2017.

*“Above all, video games are meant to just be one thing: fun. Fun for everyone.*”

- Satoru Iwata

# Agradecimentos

Gostaria de agradecer à toda minha família e àqueles que sempre me apoiaram em todas as minhas decisões da vida. Agradeço aos meus pais, Antônio e Glauciérica que me educaram e comemoram essa tanto quanto a mim. Agradeço ao meu irmão Édipo que é o meu melhor amigo e sempre me acompanhou nessa vida. Também sou grato ao meu primo Jônatas que tenho como irmão e sua esposa Mayanna por torcerem pelo meu sucesso a todo momento. Sou grato pelo meu intercâmbio que expandiu meus horizontes sobre o mundo e me fez conhecer os marinheiros da Barca que fizeram essa experiência ser inesquecível. Agradeço aos professores do Centro de Informática da UFPE, que foram parte essencial da minha formação. Por fim, gostaria de agradecer especialmente ao Doutor Geber Ramalho, que me orientou durante o desenvolvimento desse trabalho, e ao Doutor Marco Túlio, que tutorou bastante sobre jogos e foi imprescindível para a conclusão da minha graduação. Agradeço também a você que está lendo esse trabalho por se interessar no assunto e espero poder corresponder sua expectativa com a minha pesquisa.

# Resumo

## FERRAMENTA DE PRÉ-ANÁLISE PARA AUXILIAR O DESIGNER NO BALANCEAMENTO DE JOGOS

Autor: Eric Ândersen Freitas e Silva
Orientador: Geber Lisboa Ramalho
Co-orientador: Marco Túlio Caraciolo Ferreira Albuquerque

*Em jogos precisamos balancear as regras e parâmetros para que seja divertido jogar. Dizemos que um jogo competitivo é balanceado quando não há estratégia dominante sobre as outras estratégias. Para atacar esse problema propusemos utilizar aprendizagem de máquina para simular jogadores em busca de uma estratégia dominante. Em nossa ferramenta construída utilizamos de aprendizagem por reforço em cada iteração explorando as regras do jogo com o propósito de ganhar. Através de iterações feitas na nossa ferramenta foi possível que um game designer avalia-se se o jogo encontrava-se balanceado ou não. Nossos resultados mostram como é possível encontrar as falhas de balanceamento através da ferramenta e ainda modificar os parâmetros que resultem em um jogo balanceado mais rapidamente e confiavelmente.*

**Keywords:** inteligência artificial, aprendizagem por reforço, jogos, balanceamento de jogos, jogos competitivos, jogos online

# Abstract

## PRE-ANALYSIS TOOL TO ASSIST THE DESIGNER IN GAMING BALANCING

Author: Eric Ândersen Freitas e Silva
Advisor: Geber Lisboa Ramalho
Co-advisor: Marco Túlio Caraciolo Ferreira Albuquerque

*In games we need to balance rules and parameters in order for it to be fun. We state that a competitive game is balanced when there is no dominant strategy over the other strategies. To attack this problem we proposed to use machine learning to simulate players searching for a dominant strategy. In our built tool we used reinforcement learning in each iteration exploiting the rules of the game with winning as sole purpose. Through iterations made in our tool it was possible for a game designer to evaluate whether the game was balanced or not. Our results show how it is possible to find balancing problems through the tool and also change its parameters that result in a balanced game quickly and trustfully.*

**Keywords:** artificial intelligence, reinforcement learning, games, game balancing, competitive games, online games

# Lista de ilustrações

# Sumário

# 1 Introdução

## 1.1 Motivação

Jogos hoje em dia estão bem mais complexos e acessíveis do que antigamente. Ao mesmo tempo que existem ferramentas que facilitam pequenos desenvolvedores a criarem jogos com uma qualidade superior à décadas passadas grandes estúdios investem milhões de dólares em um único projeto. Mesmo com essa diferença de capital os jogos em sua essência devem ser divertidos aos jogadores. Para entreter esse jogador o jogo deve demonstrar desafios que não podem ser difíceis demais para não o frustrar nem fáceis demais para não o entediar. [19][20]

Para criar desafios apropriados à um jogador é necessário balancear as regras e os parâmetros do jogo para evitar o descontentamento de quem joga. Balancear um jogo é fazer com que que existam diversos caminhos para a vitória e que nenhum desses caminhos seja uma estratégia superior à todas as outras. [2]

Num estudo feito em Stanford demonstrou que jogadores são motivados por analisar e entender mecânicas e por competir contra outros jogadores para demonstrar poder. Jogos que possuem competições entre jogadores conseguem aliar essas duas motivações. Jogadores investem tempo para entender as regras do jogo e formular estratégias, e esperam ser recompensados pela aplicação correta dessas estratégias. O problema é que jogos que possuem competição são mais sensíveis ao balanceamento por causa da expectativa dos jogadores de que as regras do jogo são estáticas.

Mesmo no começo do desenvolvimento é necessário que já exista um balanceamento em jogos competitivos para que ele seja mais divertido, mas técnicas atuais de balanceamento não são efetivas para eles. Por exemplo, em balanceamento automático o foco é alterar regras e estratégias utilizadas dinamicamente para que a partida seja mais equilibrada do ponto de vista do

sistema, mas isso vai de encontro com a motivação do jogador já que as regras estão mudando após a formulação de sua estratégia e  graças ao balanceamento dinâmico o jogador que possui um domínio maior das regras não se sentirá recompensado por ter investido seu tempo entendendo as regras do jogo. Por isso o jogo precisa ser configurado previamente e ser justo para que em um conflito entre jogadores alguém não se sinta roubado de sua vitória por causa de alguma regra que não foi pensada pela equipe de desenvolvimento ou que foi alterada durante o combate. [9][18][19]

Em MMORPGs (*Massively multiplayer online role-playing games*) podemos criar personagens inerentemente diferentes uns dos outros pelo estilo e regras do jogo ao através da escolha da raça, a classe e poderes do personagem jogável. Essas escolhas garantem que um personagem não será igual à outro personagem jogado por outro indivíduo, por exemplo as ações de um mago dificilmente são disponíveis em outra classe de personagem e mesmo que exista os parâmetros dessa ação são alterados para que exista uma variação entre os poderes. [2]

O balanceamento de um MMORPG é importante pois jogos competitivos são mais atrativos quando possuem um equilíbrio de estratégias. Um jogo que não possui este equilíbrio se torna desinteressante e enfadonho. Os jogadores que desejam competir deverão utilizar uma estratégia dominante para terem uma chance de ganharem e quem não a utilizar se sentirá roubado de uma vitória. Já um jogo balanceado prende um jogador por ser um sistema justo onde um jogador deve combater o outro e àquele que possuir a maior habilidade e entendimento das regras do jogo sairá vitorioso. [2][8]

Com isso vemos que para balancearmos um jogo competitivo precisamos que um *game designer* tenha uma maneira de prever essas estratégias dominantes dentro de um conjunto de regras criadas. [2][18]

## 1.2 Objetivos

Neste trabalho propusemos criar uma ferramenta em que será dada como entrada as regras do jogo e as ações de dois personagens jogáveis e será produzido um

conjunto de documentos com *logs* e estatísticas que auxiliará o *game designer* a balancear o jogo ainda na fase de produção do jogo.

## 1.3 Abordagem

Através de aprendizado de máquina é possível que um computador explore as ações que cada personagem pode realizar em busca de uma estratégia dominante. Utilizar algoritmos de inteligência artificial que tenha uma natureza exploratória é essencial para encontrar a estratégia dominante na instância de regras passadas.

## 1.4 Estrutura do documento

Os capítulos a seguir destrincham o trabalho em:

Capítulo 2 - Balanceamento em Jogos: Neste capítulo iremos explorar os problemas em tentar balancear um jogo e quais as necessidades de uma solução que busca resolver esse problema

Capítulo 3 - Técnicas de Balanceamento: Este capítulo mostra analiticamente quais as técnicas utilizadas atualmente.

Capítulo 4 - Proposta de Solução: Neste capítulo mostramos a solução proposta inspirada nas técnicas atuais.

Capítulo 5 - Estudo de Caso:  Aqui mostramos a implementação da solução que foi proposta aplicada à um jogo em desenvolvimento.

Capítulo 6 - Resultados: O capítulo que compila os resultados alcançados ao mostrar as iterações realizadas com a ferramenta que foi proposta e uma análise desses resultados

Capítulo 7 - Conclusões: Neste capítulo discutimos a eficácia e as limitações da ferramenta, e sugerimos trabalhos futuros como melhoria desta técnica.

# 2 Balanceamento em Jogos

## 2.1 O problema

Balancear um jogo é uma atividade que primeiramente requer que perguntemos o que é um jogo balanceado. Um jogo balanceado às vezes pode ser atingido pela simetria de sistemas. Não é o caso de MMORPGs já que a assimetria de mecânicas entre os personagens é o fator que atrai jogadores. Existe também a diferença de balanceamento entre um conteúdo cooperativo onde os jogadores colaboram para conquistar um desafio imposto pelo jogo, à isso é dado o nome de PvE (*Player-versus-Environment*), e entre um conteúdo competitivo onde os jogadores devem competir entre si num duelo para ver quem é o mais habilidoso, o PvP (*Player-versus-Player*). [1] [13]


Figura 1. Jogo de luta Street Fighter

O balanceamento difere bastante por causa das regras envolvidas em um jogo e o número de interações que ocorrem entre estas regras. Em um jogo de tabuleiro como Damas é possível dizer que o balanceamento existe se nenhum dos jogadores possui uma vantagem clara por começar com as peças brancas ou peças pretas, mas já para jogos de luta como *Street Fighter* o balanceamento pode vir do fato de alguns personagens serem mais efetivos contra outros oferecendo uma vantagem para aquele que souber quem é efetivo contra qual outro personagem.

Para alguns jogos é possível provar que ele é matematicamente balanceado, mas esse não é o caso para a maioria dos jogos por serem complexos demais e por isso atualmente ainda é feito um balanceamento através de sessões de *playtest* onde pessoas jogam o jogo e suas ações e os resultados de suas ações são arquivados para que os *game designers* do jogo possam analisar se o jogo está balanceado ou não. [2]

Esta tarefa acaba se tornando um empecilho para o desenvolvimento do jogo já que não é possível prever quanto tempo levará ou quanto custará para balancear o jogo. Em casos de MMORPGs os designers não conseguem garantir a ausência de uma estratégia dominante já que as sessões de *playtesting* podem não ser suficiente para explorar todas as possibilidades pois isso necessitaria de muito tempo. [8] Adicionalmente mesmo que uma estratégia dominante seja identificada as regras teriam de ser modificadas para balancear o jogo e o processo seria iniciado novamente. Além disso *playtests* feitos para balancear um jogo é uma operação não escalável. [1] E mesmo que um jogo já esteja balanceado a adição de regras ao sistema acarretará na necessidade de uma nova verificação por uma estratégia dominante que podem desbalancear o jogo já que é difícil de se avaliar o impacto indireto da adição de uma regra em todos os outros aspectos do jogo. [8]

## 2.2 Requisitos de uma solução

Uma alternativa mais recente à esse problema é o uso de ferramentas que fazem análises automáticas em busca da resposta para se o jogo é ou não balanceado de forma estática. Assim como é preciso identificar o que é o estado de balanceamento ótimo de um jogo também é necessário explorar as alternativas para uma análise do jogo. Jogos PvE e PvP possuem necessidades diferentes que a ferramenta deve atender ao processar os dados. Para MMORPGs onde o foco é o PvP a solução deve explorar as regras e suas interações em busca de uma estratégia dominante. [2] [5]

As regras do jogo serão inseridas na ferramenta num processo de configuração de ambiente onde variáveis do sistema de regras deve ser descrita de

forma que esses parâmetros possam ser alterados facilmente no futuro seja de forma automática ou manual. Em seguida será necessário criar uma função objetivo que utiliza todos os valores atuais do estado e representa numericamente o quão bem aquele estado é sem depender de estados anteriores. Esta função deve ser escolhida cuidadosamente visto que ela é a representação do que é balanceamento do jogo analisado. Por fim será escolhido o algoritmo que buscará a estratégia dominante. Uma vez decidido começa o processo automático de simulação. [14]

Nesta etapa o algoritmo escolhido simula uma sessão o jogo tentando utilizar a função objetivo determinada para explorar as regras do jogo. Para todas as simulações os dados são armazenados para que esses dados possam ser avaliados no futuro. Aqui faz-se uso de um sistema inteligente que representará os jogadores tomando as decisões. Estas simulações continuarão sendo otimizadas a cada iteração até que um critério de parada previamente definido seja alcançado ou um tempo limite se esgote. [14]

Essa solução deve ser mais rápida e intuitiva de iterar do que as soluções atuais. O tempo e esforço gasto pelo *game designer* devem ser reduzidos pelas informações que vão ser-lhe oferecidas pela ferramenta. [1]

# 3 Técnicas de Balanceamento

Visto que balancear um jogo é importante para o sucesso do jogo alguns estudos já foram feitos nessa área. Desenvolvedoras também possuem seus próprios métodos e ferramentas de balanceamento e embora sejam ferramentas proprietárias elas são inspiradas nas técnicas abaixo e são trabalhadas para que provejam melhores resultados de acordo com as necessidades do jogo. [15]

## 3.1 Experimental Design

É o método de balanceamento mais utilizado. É através do talento do *game designer* e testes com humanos que o jogo se balanceia. Ele é um processo iterativo que necessita que humanos testem o jogo e tenham todas as suas atividades armazenadas. Ao fim da sessão de jogo os testadores respondem questionários e participam de entrevistas para coletar dados que não foram pegos pelo jogo. [1]

É uma técnica cara, pode demorar e não é fácil encontrar um *game designer* talentoso. Tem como vantagens a natureza evolutiva das regras do jogo e a possibilidade de captar nuances que normalmente não são percebidas em testes feitos pelos desenvolvedores. [1]

## 3.2 Restricted Play

Restricted Play é uma técnica desenvolvida na University of Washington que tem como objetivo facilitar o processo de balanceamento através de um processo que deve ser seguido. [8]

O sistema propõe uma série de perguntas que deve ser respondida para cada parte do sistema de regras que se deseja balancear e a partir de cálculos matemáticos são feitos diversas alterações seguidas de sessões do jogo para dar

como entrada de novo ao sistema. A cada iteração uma regra é demonstrada como problemática seja por ser uma estratégia fraca demais ou forte demais. Porém assim como o Design Experimental é necessário o uso de teste humano e de talento de um *game designer* para planejar as estratégias. Além disso ela não é capaz de captar estratégias que não foram pensadas pela equipe de desenvolvimento. [8]

## 3.3 Técnicas com algoritmos genéticos

Um artigo de 2008 já mostra o uso de algoritmos com inteligência artificial para balancear jogos. O uso de um algoritmo que busca por uma estratégia dominante é uma alternativa mais barata e rápida já que não exige interação com humanos. [2]

Para o desenvolvimento dessas ferramentas é necessário escolher o algoritmo que será utilizado que simulará os jogadores aplicando certas estratégias. Num algoritmo feito na University of Nevada foi utilizado Agentes Autônomos para simular as possíveis estratégias a serem empregadas. Cada estratégia foi implementada em um agente e eles se revezavam à medida que era necessário após observar o cenário do jogo em busca da vitória. Após algumas iterações já é possível ver se existe alguma estratégia dominante [2]

## 3.4 Balanceamento Dinâmico

Em balanceamento dinâmico os desafios apresentados ao jogador são nivelados para que ele seja sempre adequado a suas habilidades. Com isso um jogador será mantido sempre num estado onde ele não estará entediado nem frustrado.

O balanceamento se dá pelo perfil do usuário e medirá sua habilidade, conhecimento do jogo e capacidade de aprender sobre o jogo para definir quais os parâmetros dos desafios que ele enfrentará ou alterar o comportamento de agentes automáticos para se igualar ao oponente. Há portanto uma limitação nas abordagens dinâmicas sobre jogos competitivos já que não é possível alterar o comportamento do jogo durante uma partida. [18]

# 4 Proposta de solução

## 4.1 Balanceamento na Fase de Produção

A nossa solução precisar apoiar o *game designer* ainda na fase de produção do jogo e deve somente avaliar as regras atuais do jogo e não alterá-las durante a partida. Não existem métricas de dados de usuários para auxiliar na busca da estratégia dominante então por isso propusemos utilizar uma inteligência artificial que simulará os jogadores em busca da vitória. Cada uma das simulações é armazenada e um compilado dessas simulações é feito mostrando estatísticas retiradas das simulações

Em seguida, os dados gerados pela etapa automática são passados para o *game designer* que analisará estas simulações, os documentos gerados e as estatísticas e decidirá se o jogo está ou não balanceado. Ele também utilizará os dados obtidos para melhorar o processo de balanceamento sugerindo adições ou remoções de variáveis na configuração de regras e mudanças na função objetivo. Este passo inclui o aspecto de aprendizado do processo de balanceamento ao aperfeiçoar os passos anteriores. Baseado na análise do *game designer* ele decide se o processo deve voltar ao primeiro passo com as devidas alterações ou se os resultados são satisfatórios e o jogo já está balanceado. [14]

## 4.2 Aprendizado por Reforço

Visto que aprendizado de máquina é essencial por sua natureza exploratória de soluções escolhemos uma modelagem de aprendizado por reforço. A aprendizagem por reforço é ideal para o problema já que não possuímos nenhum dado prévio para treinamento ou teste por exemplo. O problema de balanceamento de um jogo competitivo também faz-se uso de uma abordagem de algoritmos genético já que os agentes devem aprender a lutar contra seu oponente em seu turno.

Em um modelo de aprendizado por reforço os agentes recebem a cada interação como entrada o estado atual dado o ambiente que gera uma ação como saída. Essa ação é aplicada ao estado atual gerando um novo estado e com isso é calculado um valor de reforço com base nessa transição realizada. O comportamento do agente então é modificado para que os pares de estado e ação que sejam mais bem avaliados sejam mais escolhidos no futuro. [6]

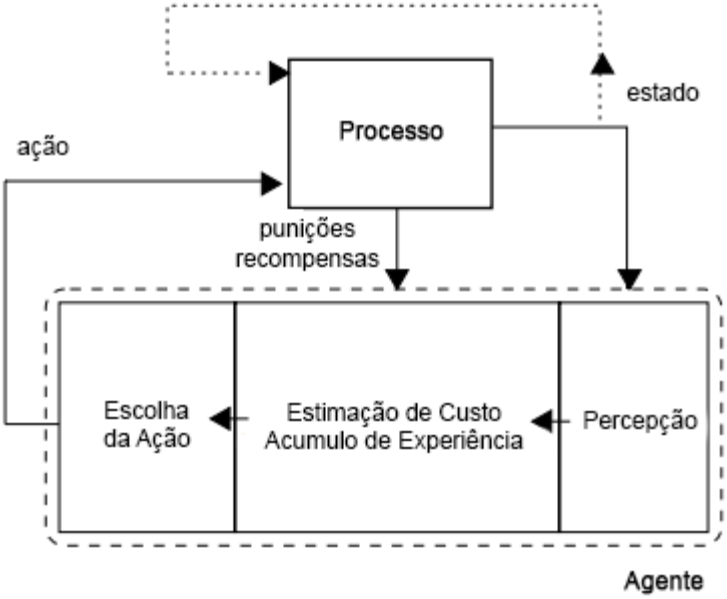


Figura 2. Interação entre agente e o processo de Aprendizado por Reforço. [3]

Entre os processos de Aprendizado por Reforço devemos escolher o algoritmo que se encarregará de como o novo estado influencia as decisões do agente. Entre as possíveis soluções escolhemos a do *Q-learning* por ter a *q-table*, uma tabela que contém o valor das ações que um agente pode tomar dado o seu estado atual, que pode auxiliar na interpretação dos dados por parte do *game designer* já que ela é mais legível do que outras estruturas de dados resultantes do processamento de outros algoritmos.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\ a_t = a,\ \pi\right]$$

Figura 3. Fórmula do Q-learning. [3]

Complementarmente o *Q-learning* é conceitualmente mais simples, mais estudado e possui mais implementações, além de ter sido mostrado empiricamente que converge à um resultado mais rapidamente do que outros algoritmos. [10] O *Q-learning* se resume à uma função que dados um estado (s) e uma ação (a) resulta na maior recompensa possível para esse estado (s) dado todas as possíveis iterações futuras do estado que será atingido após a ação (a) ser executada em no estado (s) utilizando uma política π. [16]

# 5 Estudo de Caso

## 5.1 O Estudo de caso

Para a nossa solução foi escolhido um jogo com foco em PvP que está em desenvolvimento e portanto não possui dados de jogadores especialistas reais ainda. Modelamos as regras do jogo em estados que os agentes podem alcançar e as ações que fazem eles alcançar esses estados. Foram escolhidas as duas classes mais básicas do jogo já que são simples o suficiente para serem modeladas cada uma como um agente próprio com regras diferentes. A partir daí o processo se torna uma iteração das seguintes etapas:

1. Evoluir cada agente em função de derrotar o oponente através da simulação de duelos com o aprendizado por reforço. Cada agente terá a oportunidade de evoluir suas estratégias contra o seu oponente divididas em 6 iterações dos passos a seguir.
    a. Um agente será escolhido como desafiante e o outro será posto como defensor.
    b. O desafiante passará pelo processo de aprendizado por reforço enquanto o defensor utilizará um plano entre uma lista de planos que ele já utilizou no passado e resultou na sua vitória.
    c. Após 500 iterações do passo anterior caso a condição de parada não tenha sido atingida ainda os papéis são invertidos e o passo acima é repetido.
2. Com o resultado das 3000 partidas em mão vamos analisar o resultado dos duelos em comparação com as políticas criadas pelos agentes.
3. Se o jogo estiver desbalanceado deve-se ajustar os parâmetros das regras que estão sendo uma estratégia dominante e repetir o processo do começo.

Com isso um *game designer* tem uma noção maior dos impactos que cada variável pode ter para a criação de estratégias por parte dos jogadores e pode refinar a experiência dos jogadores com um jogo mais balanceado. [2]

## 4.3 Ferramentas

Para modelar os estados e ações foi utilizado a biblioteca BURLAP (*Brown-UMBC Reinforcement Learning and Planning*) feito pela Brown University. Essa biblioteca reúne classes que dão suporte à criação de problemas próprios que vão ser resolvidos pelos diversos algoritmos de aprendizagem.

Para a construção da ferramenta com base no jogo escolhido foi necessário modelar as regras do jogo implementando para cada agente individualmente:

1. *Domain*: Aqui é onde colocamos todas as informações sobre todos os estados, ações, transições e funções de avaliação.
2. *Model*: Objeto que possui as regras do jogo e é a base para a criação de estados.
3. *State*: É a representação atual do mundo onde o agente se encontra. Aqui é onde o agente age em função dos sensores para saber qual a situação do estado e calcula a melhor ação a ser tomada.
4. *Action*: As ações possíveis do agente.
5. *StateTransition*: Uma função de estados e ações onde o agente utiliza para consultar quais os estados alcançáveis dado o estado atual e a lista de ações que o agente pode tomar neste estado. Também é utilizada para definir as probabilidades de sucesso das ações em jogos estocásticos .

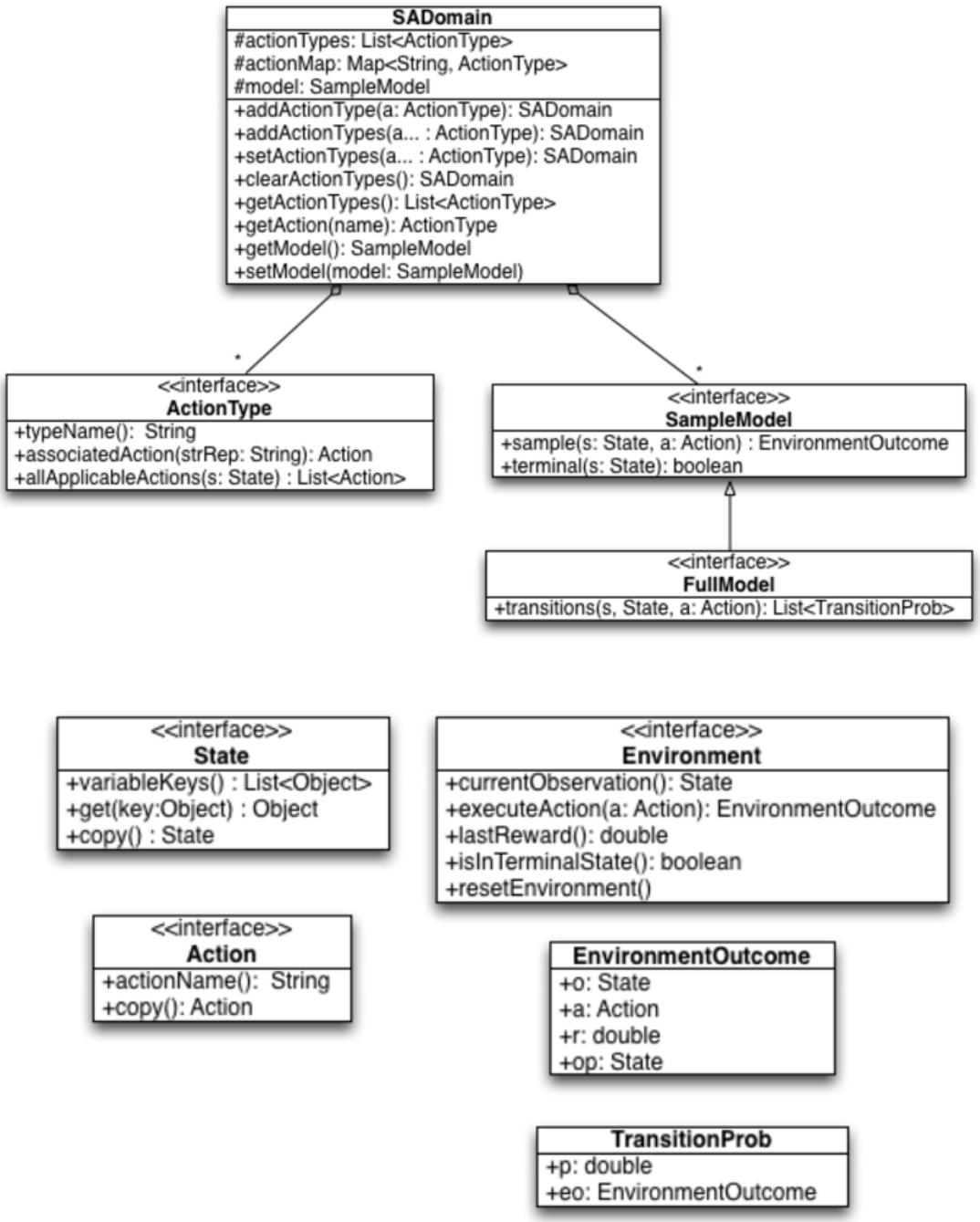


Figura 4. Diagrama de classes da UML das principais classes, interfaces e funções do BURLAP [17]

Com as regras já inseridas agora o processo precisa das funções *TerminalFunction* e *RewardFunction* que agem de maneira similar ao retornar um valor que é passado ao algoritmo para melhorar as políticas de escolha de ações dado um certo estado, mas diferem no fato da *TerminalFunction* indicar que o estado que o agente chegou é um estado terminal enquanto que *RewardFunction* não.

A *RewardFunction* recebeu uma atenção especial já que nas rodadas de teste os agentes estavam atrasando suas ações em prol de receber uma recompensa menor, mas constante ao invés de terminar a partida e ganhar uma recompensa considerável, isto é, em um momento que definimos que a saúde do seu inimigo era a recompensa do agente alguns agentes aprenderam a reduzir a saúde do inimigo rapidamente e enquanto estivessem fora de perigo preferiam atrasar derrotar o inimigo em prol de receber mais recompensa. Para resolver isso foi necessário trocar a *RewardFunction* para que o agente só recebesse o incentivo equivalente ao quanto ele conseguiu retirar da saúde do inimigo durante a transição de estados.

ql_0000.episode

```
1 !!burlap.behavior.singleagent.Episode
2 actionSequence:
3 - &id003 !!burlap.mdp.core.action.SimpleAction {name: shoot3}
4 - &id001 !!burlap.mdp.core.action.SimpleAction {name: shoot2}
5 - &id002 !!burlap.mdp.core.action.SimpleAction {name: shoot1}
6 - !!burlap.mdp.core.action.SimpleAction {name: miss}
7 - *id001
8 - *id002
9 - *id003
10 rewardSequence: [15.0, 35.0, 25.0, 15.0, 35.0, 15.0, -1000.0]
11 stateSequence:
12 - !!br.ufpe.cin.ArenaState {enemyHP: 100, heroHP: 100}
13 - !!br.ufpe.cin.ArenaState {enemyHP: 100, heroHP: 85}
14 - !!br.ufpe.cin.ArenaState {enemyHP: 80, heroHP: 70}
15 - !!br.ufpe.cin.ArenaState {enemyHP: 70, heroHP: 55}
16 - !!br.ufpe.cin.ArenaState {enemyHP: 70, heroHP: 40}
17 - !!br.ufpe.cin.ArenaState {enemyHP: 50, heroHP: 25}
18 - !!br.ufpe.cin.ArenaState {enemyHP: 50, heroHP: 10}
19 - !!br.ufpe.cin.ArenaState {enemyHP: 10, heroHP: 0}
20
```

Figura 5. Exemplo de um episódio armazenado em formato de texto.

Em seguida foi implementada a classe *LearningAlgorithm* que trata da instanciação do *Domain*, *States*, e a implementação da função que garante o aprendizado, no caso o *Q-learning*. É nessa classe que o agente roda em forma de um *LearningAgent* e executa um episódio por vez. Cada um desses episódios são armazenados em formato de texto.

# 6 Resultados

## 6.1 Iterações

Em todos as iterações de aprendizado foram feitos 1500 episódios de aprendizado para cada um dos dois agentes e eles teriam de tentar eliminar o adversário o mais rápido que pudessem. Somente os últimos 100 episódios eram utilizados como parte da mineração de dados que buscava a estratégia dominante.

A primeira iteração foi feita com as regras que foram recebidas inicialmente sem quaisquer alterações. Nela o espadachim ganhou em 71 das 100 últimas partidas contra o mago. Na revisão foi visto que o espadachim conseguia diminuir rápido demais a saúde do mago com golpes bem mais precisos. O *game designer* analisou e notou que a partidas acabam com uma média de 3,2 rodadas e isso não era o desejado para o jogo. Havia uma expectativa que as partidas durassem mais para que houvesse mais oportunidades de estratégias de defesa utilizando o cenário do jogo. Para alcançar esse objetivo o dano das armas duas classes foram reduzidos.

Com o resultado da segunda iteração de aprendizado foi visto que agora o espadachim ganhava 43% das partidas. Com a análise dos episódios notou-se que 41 das 57 partidas em que o mago ganhou ele utilizava o seu golpe especial que encerrava a partida mesmo quando o espadachim estava ganhando. O especial do mago estava possuía uma taxa de acerto alta demais para o dano que causava e por essa razão foi escolhido um novo valor do dano do especial que faria com que o espadachim sobrevivesse em mais sete dessas partidas que ele havia perdido.

A terceira iteração resultou em 48% das partidas ganhas pelo espadachim com uma média de 5,4 rodadas por partida. O resultado agradou ao *game designer* e essas mudanças foram integradas ao jogo. Para dar continuidade com esse balanceamento seria necessário agora avaliar as próximas classes que não foram

balanceadas ainda para que elas sejam balanceadas contra essas duas classes que já foram testadas.

## 6.2 Análise

Inicialmente houveram dúvidas se de fato o agente do espadachim estava aprendendo a combater o mago com o algoritmo e para isso foram analisados os 100 primeiros episódios da primeira iteração onde foi observado que o percentual de vitórias era de 42%. O número de vitórias parece ter sido grande para um agente que ainda estava aprendendo quais são as melhores estratégias, mas é compreensível quando o número de escolhas é pequeno. Essa porcentagem só reforça o como o espadachim já estava mais poderoso do que o mago já que mesmo sem uma estratégia bem definida o espadachim conseguia ganhar algumas partidas.

O resultado da primeira iteração foi interessante pois despertou a atenção do *game designer* para um aspecto de *gameplay* que não havia sido levado em consideração na definição inicial das regras. A partida estava acabando rápido demais para que uma estratégia pudesse fazer qualquer efeito e a sensação do jogador de que ele precisa aprender o jogo seria limitada por isso fazendo o jogo ser mais raso a retenção de jogadores seria menor por causa disso. [4]

Após a reconfiguração das regras a segunda iteração do experimento resultou em uma drástica mudança dos resultados que revelaram que a configuração do mago estava fora do padrão. Foram sugeridas algumas possibilidades do que fazer para melhorar as chances do espadachim como por exemplo alterar a o dano das habilidades do mago para que seu especial fosse mais difícil de virar o combate, mas com a análise mais profunda percebeu-se que o especial do mago era justamente a causa do espadachim ter a vitória roubada mesmo em um jogo que parecia estar na vantagem. A quantidade que precisou ser diminuída também foi retirada diretamente da análise o que facilitou na escolha do novo dano do especial.

# 7 Conclusões

## 7.1 Discussão

A ferramenta traz uma abordagem nova para uma área que está em desenvolvimento ao permitir um balanceamento do jogo ainda na fase de produção. Ela também é focada para jogos competitivos entre jogadores um foco que não há tantos trabalhos. Ainda existem melhorias que precisam ser melhores trabalhadas, mas é clara a melhoria trazida pela ferramenta em termos de velocidade do processo e acurácia do balanceamento.

Adicionalmente regras inteiras podem ser testadas e refinadas antes mesmo de serem codificadas no jogo já que é mais simples de modelá-las na ferramenta do que implementar todo o sistema no jogo em desenvolvimento e fazer com que as pessoas testem da mesma forma que o balanceamento seria feito.

Houve dois aspectos que ficaram insatisfatórios quanto à ferramenta. A primeira é que embora o processo de modelar as regras de um jogo não é transferível entre projetos. Para o balanceamento de um novo jogo é difícil de se reutilizar de códigos já prontos a menos que os jogos possuam regras já muito parecidas. A adição de algumas regras novas também podem exigir o retrabalho de tudo que foi feito até então para que a regra nova seja inserida

A segunda insatisfação é que a única maneira que tratamos de máximos locais na hora de pesquisar pelas estratégias é utilizar alguma estratégia não tão eficiente em alguns momentos, mas isso pode não surtir efeito se o número de iterações não for grande o suficiente e as partidas forem longas o bastante.

## 7.2 Trabalhos Futuros

A ferramenta funcionou bem para um jogo onde as regras ainda eram simples o suficientes, mas algumas melhorias seriam necessárias caso o jogo fosse diferente e em alguns casos seria necessário uma ferramenta nova que comportaria as necessidades do novo jogo.

1. Facilitar o processo de modelagem das regras

Ainda é necessário alguém com um conhecimento técnico para modelar as regras na ferramenta, mas com um esforço voltado em categorização e criação dessas regras é possível que o próprio *game designer* consiga utilizar a ferramenta sem a necessidade de ter que explicar para um terceiro as regras que ele tem em sua mente.

2. Automatizar o processo de ajuste dos parâmetros

Na nossa ferramenta ainda é necessário o conhecimento do *game designer* para manualmente analisar e fazer as alterações do jogo, mas um algoritmo suficientemente robusto poderia fazer esse processo automaticamente dado um certo ponto de parada e uma indicação de quais parâmetros ele poderia editar. Com um algoritmo de aprendizado ele poderia interpretar qual parâmetro influencia negativamente ou positivamente certos resultados e poderia convergir para um jogo balanceado sozinho. [2]

3. Visualização

A maneira como é entregue os documentos para análise é bem crua. Diagnosticar qual era o problema de balanceamento foi certamente mais fácil do que o normal, mas poderia ter sido melhor se mais tempo tivesse sido investido em maneiras de mostrar esses resultados. Gráficos e um modo de rejogar os episódios das partidas seria uma maneira mais intuitiva de visualizar o que está acontecendo com o jogo. [2]

4. Balanceamento para jogos com muitas informações ocultas ou com sessões muito longas

O *Q-learning* utilizado na ferramenta não consegue lidar com estados onde muitas das informações necessárias para uma tomada de decisão são escondidas. Além disso um jogo com uma duração muito longa deve fazer o algoritmo demorar bastante para que ele entenda quais as melhores ações a se tomar visto que uma ação só terá impacto num futuro incerto e por isso haverá problemas em convergir em uma estratégia dominante. [8][16]

# Referências Bibliográficas


[1] CHEN, Haoyang; MORI, Yasukuni; MATSUBA, Ikuo. Solving the balance problem of on-line role-playing games using evolutionary algorithms. **Journal of Software Engineering and Applications**, v. 5, n. 08, p. 574, 2012.

[2] LEIGH, Ryan; SCHONFELD, Justin; LOUIS, Sushil J. Using coevolution to understand and validate game balance in continuous games. In: **Proceedings of the 10th annual conference on Genetic and evolutionary computation**. ACM, 2008. p. 1563-1570.

[3] RIBEIRO, Carlos Henrique Costa. A tutorial on reinforcement learning techniques. In: **Supervised Learning Track Tutorials of the 1999 International Joint Conference on Neuronal Networks**. 1999.

[4] ROLLINGS, Andrew; MORRIS, Dave. Game architecture and design: a new edition. 2003.

[5] SHUI, Linlin et al. Game Balance Principles in MMORPG with Pet System. **Learning by Playing. Game-based Education System Design and Development**, p. 133-140, 2009.

[6] KAELBLING, Leslie Pack; LITTMAN, Michael L.; MOORE, Andrew W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, v. 4, p. 237-285, 1996.

[7] MERRICK, Kathryn; MAHER, Mary Lou. Motivated reinforcement learning for non-player characters in persistent computer game worlds. In: **Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology**. ACM, 2006. p. 3.

[8] JAFFE, Alexander et al. Evaluating Competitive Game Balance with Restricted Play. In: **AIIDE**. 2012.

[9] YEE, Nicholas. Motivations of play in MMORPGs. 2005.

[10] SUTTON, Richard S.; BARTO, Andrew G.; WILLIAMS, Ronald J. Reinforcement learning is direct adaptive optimal control. **IEEE Control Systems**, v. 12, n. 2, p. 19-22, 1992.

[11] JAFFE, Alexander Benjamin. **Understanding game balance with quantitative methods**. 2013. Tese de Doutorado.

[12] VOLZ, Vanessa; RUDOLPH, Günter; NAUJOKS, Boris. Demonstrating the Feasibility of Automatic Game Balancing. In: **Proceedings of the 2016 on Genetic and Evolutionary Computation Conference**. ACM, 2016. p. 269-276.

[13] SCHELL, Jesse. **The Art of Game Design: A book of lenses**. CRC Press, 2014.

[14] BEYER, Marlene et al. An Integrated process for game balancing. In: **Computational Intelligence and Games (CIG), 2016 IEEE Conference on**. IEEE, 2016. p. 1-8.

[15] **Developer Update | Matchmaking & Hero Balance | Overwatch.** Blizzard Entertainment, 2017. 9 minutos. Disponível em: “https://www.youtube.com/watch?v=04a7Mv_eWdY”. Acesso em: Julho de 2017.

[16] MNIH, Volodymyr et al. Human-level control through deep reinforcement learning. **Nature**, v. 518, n. 7540, p. 529-533, 2015.

[17] BURLAP. Disponível em: <http://burlap.cs.brown.edu/>. Acesso em 4 de julho de 2017.

[18] DANZI, Gustavo et al. Online adaptation of computer games agents: A reinforcement learning approach. In: **II Workshop de Jogos e Entretenimento Digital**. 2003. p. 105-112.

[19] EGENFELDT-NIELSEN, Simon; SMITH, Jonas Heide; TOSCA, Susana Pajares. **Understanding video games: The essential introduction**. Routledge, 2015.

[20] CHICKZENTMIHALYI, Mihaly. The psychology of optimal experience. 1990.