# Usando IA para realizar ajustes de dificuldade dinâmicos em um sistema de batalha em turnos

**Douglas Vanny Bento**[1]

[1]FACIN/FAMECOS – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

douglas.bento@acad.pucrs.br

***Abstract.*** *This paper analyses the development of a Dynamic Difficulty Adjustment in a simultaneous turn-based battle system. We detail the implementation of an artificial intelligence that is able to interpolate between different levels of difficulty and then we approach multiple criteria to define when is there a need to make adjustments in the game's difficulty for different players.*

***Resumo.*** *Esse artigo analisa o desenvolvimento de um sistema de ajuste dinâmico de dificuldade (Dynamic Difficulty Adjustment) em um sistema de batalha baseado em turnos simultâneos. Detalhamos a implementação de uma inteligência artificial capaz de interpolar entre níveis de dificuldade diferentes e abordamos diferentes critérios para definir quando há a necessidade de realizar ajustes da dificuldade do jogos para diferentes jogadores.*

## 1. Introdução

A indústria de jogos está entre as de maior faturamento numa escala global, superando a industria de filmes e de música. Enquanto desenvolvedores de jogos, é sempre importante se preocupar com a experiência que jogadores de diferentes níveis de habilidade irão ter ao experiênciar o jogo. Nesse ponto, o controle de dificuldade se torna crucial.

Ao longo desse artigo falaremos sobre dificuldade em jogos e formas de manipular a mesma para jogadores de diferentes níveis de habilidade, usando como domínio o jogo *Capital* [Bento et al. 2018]. Na sessão 2 exploraremos o conceito de *Flow* e sua ligação extremamente forte com jogos. A seguir introduziremos o leitor ao conceito de *Dynamic Difficulty Adjustment* (Ajuste dinâmico de dificuldade) e sua ligação com a teoria de *Flow* na sessão 3. Na sessão 4 apresentaremos o jogo utilizado como domínio no estudo realizado, explicando ao longo da sessão 5 a inteligência artificial desenvolvida para o estudo. Chegando na sessão 6, detalharemos nossa implementação, passando por suas possibilidades de manipulação e controle por parte de um Game Designer. Avaliamos nossos resultados na sessão 7 e então concluímos o trabalho na sessão 8, sitando possibilidades de trabalhos futuros.

## 2. Dificuldade e Flow

Jogos eletrônicos são uma das poucas formas de mídia eletrônica na qual o usuário tem o poder de interagir com a obra que está consumindo. Isso acaba por gerar uma diferença crucial quando comparada a diversas outras formas de mídia: a forma como a pessoa interage com a obra em si. Quando estamos assistindo a um filme, ouvindo uma música ou lendo um livro, estamos essencialmente apenas absorvendo o conteúdo. Podemos dizer

que consumir aquele conteúdo é o objetivo intrínseco de uma obra de entretenimento em si. Por este motivo, quanto menor for a dificuldade da pessoa em consumir a obra, melhor tende a ser o resultado dela.

Quando se tratam de jogos, porém, a dificuldade deixa de ser simplesmente um obstáculo e passa a ser parte do objetivo pelo qual jogamos. [Salen et al. 2004] trabalha em cima das diferentes definições de o que é um jogo. Para esse artigo, vamos considerar jogos como sistemas onde o usuário executa uma ou várias atividades a fim de atingir objetivos propostos pelo mesmo. Tendo isso em vista, quando pensamos em jogos como meios de entretenimento, o objetivo de um jogo se torna mais claro: entreter o jogador.

De acordo com [Csikszentmihalyi 1990], o conceito de *Flow* busca explicar qual é a melhor forma de se experimentar algo. Como descrito em pesquisas anteriores, grande parte delas encabeçadas por Csikszentmihalyi, uma pessoa entra em um estado de *flow* quando sua percepção de habilidade está em perfeito balanço com a sua percepção de desafio (Figura 1). [Nakamura and Csikszentmihalyi 2014] desenvolve isso, propondo um modelo com 8 canais diferentes baseados nesse balanço (Figura 2). O novo diagrama demonstra que, além de exigir um balanço entre habilidades e desafio, é necessário ainda que ambos estejam acima da média percebida pelo usuário. Caso ambos estejam abaixo, o resultado é o completo oposto, levando o usuário a um estado de apatia [Nakamura and Csikszentmihalyi 2014]. Testes realizados por [Chanel et al. 2008] demonstram os diferentes estados de ansiedade, tédio e flow quando jogadores de diferentes habilidades jogaram o mesmo jogo de Tetris com níveis de dificuldade variados.

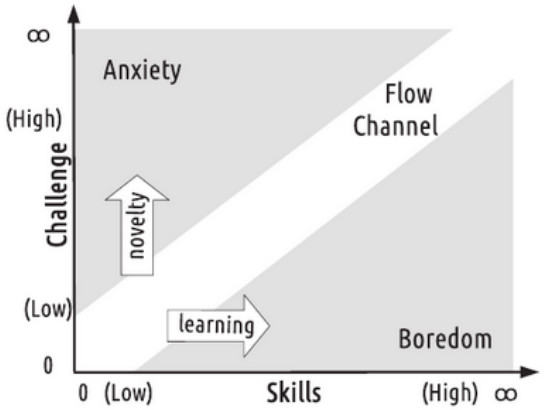


**Figura 1. Canal de *Flow*, como descrito originalmente em [Csikszentmihalyi 1990] e reproduzido por [Cruz and Uresti 2017]. O diagrama demonstra a necessidade de equilíbrio entre os níveis de habilidade e desafio. Níveis balanceados mantêm o jogador no canal de *Flow*, enquanto uma quantidade maior de habilidade ou desafio põem o jogador em um estado de Tédio ou Ansiedade, respectivamente.**

Tendo então como objetivo maximizar o estado de flow dos jogadores de um jogo, se torna necessário um cuidado muito grande com os níveis de dificuldade. [Smeddinck et al. 2016] estudou a percepção dos jogadores referentes a diferentes formas de seleção de dificuldade, demonstrando a importância na capacidade do sistema se adaptar a jogadores com maiores ou menores habilidades.

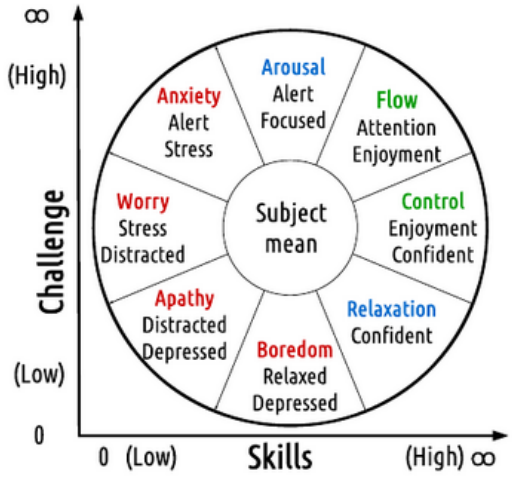


**Figura 2. Modelo de oito canais de *Flow*, como descrito originalmente em [Nakamura and Csikszentmihalyi 2014] e reproduzido por [Cruz and Uresti 2017]. O diagrama representa uma evolução do canal de *Flow* originalmente descrito na Figura 1. Além do proposto anteriormente, esse novo modelo identifica um estado de apatia quando tanto habilidade quanto desafio estão baixos, além de listar 4 estados intermediários.**

## 3. Dynamic Difficulty Adjustment

*Dynamic Difficulty Adjustment* (DDA) se refere a sistemas de ajuste de dificuldade dinâmica, muito comumente presente em jogos digitais. Esses sistemas buscam analisar a performance do jogador em um jogo e aplicar ajustes de balanço nos diversos mecanismos disponíveis. Um exemplo de DDA muito aplicado em jogos de corrida é o chamado *rubber banding*. Nesse exemplo, o jogo pode ajustar a velocidade dos carros adversários dependendo da performance do jogador na corrida, buscando oferecer uma experiência balanceada e pouco frustrante para o jogador [Cechanowicz et al. 2014].

Sistemas de DDA são extremamente sensíveis ao domínio, ao tipo de jogo ao qual está sendo aplicado, e por esse motivo suas abordagens variam consideravelmente. Todo DDA depende de uma análise de dados consistente, baseando-se em estatísticas ou dados do jogador como pontuações, geralmente interligados a uma heurística interna que avalia o quão bem o jogador está jogando [Cruz and Uresti 2017]. Todos esses dados coletados são então usados para balancear a experiência do jogador, e as diferenças entre diferentes estratégias e gêneros de jogos se acentuam nessa fase. Alguns jogos aplicam esses dados diretamente a variáveis e multiplicadores dos inimigos [Sutoyo et al. 2015, Smeddinck et al. 2016], outras abordagens trabalham em conjunto com inteligencias artificiais para ajustar variáveis no comportamento dos inimigos [Hunicke 2005] ou para auxiliar na geração de conteúdo procedural [Jennings-Teats et al. 2010], e em alguns casos essas informações são usadas para alimentar diretamente um algoritmo de aprendizado de máquina [Andrade et al. 2006].

Um ponto importante no uso de DDA é o risco do jogador perceber a dificuldade se ajustando. Em experimentos realizados por [Smeddinck et al. 2016] com jogadores testando diversas formar de selecionar dificuldade, se constatou que, de 15 jogadores que experiênciaram e perceberam o ajuste de dificuldade dinâmico, 6 se expressaram de forma bem negativa. Isso retrata a importância de gerenciar a percepção de habilidade e desafio


**Figura 3. Capturas de tela do jogo *Capital*. À direita temos uma imagem da batalha, onde o jogador deve selecionar qual habilidade usar, mostrando seu saldo e o saldo do inimigo. À esquerda temos uma imagem do final da batalha à direita. O saldo original do inimigo agora virou uma recompensa, e visto que o jogador gastou uma quantia menor a essa, ele terminou a batalha no lucro.**

do jogador sem que ele pense ser manipulado, assim tirando ele do canal de *flow*.

A fim de ter um controle sobre a dificuldade de um jogo sem que o jogador perceba essas variações, o ideal é trabalhar em cima de valores que o jogador não consegue mensurar. Uma das formas de trabalhar com isso é em cima da ideia de aleatoriedade, hoje bastante presente em alguns jogos. Para exemplificar isso nesse artigo, usaremos um jogo com batalhas em turno, no qual o jogador não tem meios de prever com 100% de precisão a ação que um inimigo ira tomar, devido a natureza aleatória de sua árvore de decisões.

## 4. Domínio

Para a escrita desse artigo, foi usado o jogo *Capital*, representado na Figura 3. Seu sistema de batalha é composto da seguinte forma. Dois personagens, um controlado pelo jogador e outro pelo computador, dispõem de uma quantia de pontos de vida que precisam manter acima de zero para que fiquem vivos. Para diminuir a quantidade de pontos de vida, cada personagem usa habilidades que tem um custo, e esse custo é reduzido de seu próprio saldo de pontos de vida.

Ao início de um turno, ambos os personagens escolhem suas ações simultaneamente. Essas ações são executadas apenas após ambos os personagens selecionarem suas ações, execução então que acontece numa ordem arbitraria. Após esse passo, o sistema então checa pelo estado de derrota, decidindo entre vitória, derrota ou empate para ambos os personagens caso algum deles não esteja mais com uma quantidade de pontos de vida positivo.

Ao fim de uma batalha, caso o personagem do jogador seja vitorioso, ele é então recompensado com o saldo que o inimigo originalmente início a batalha. Exemplificando, digamos que o jogador inicia uma batalha com 5000 pontos, contra um adversário que inicia ela com 3000 pontos. Se, ao fim da batalha, ele ainda tiver 2500 pontos restantes, os 3000 originais do inimigo são somados a essa quantia, e então o jogador sai da batalha com um total de 5500 pontos, um ganho de 10% quando comparado ao seu saldo original. Por outro lado, caso o jogador termine essa batalha com 1000 pontos, a soma dos demais 3000 resultara em apenas 4000 pontos, assim resultando num prejuízo de 20% quando comparado ao seu saldo original.

Cada inimigo tem uma série de árvores de decisão atreladas a si. Cada uma dessas árvores é composta de nodos de condição e nodos de habilidade, esses últimos sendo sempre folhas da árvore. Os nodos de habilidade tem pesos atrelados. Cada busca na árvore inicia pela raiz, e expande a partir de todos os nodos de condição que passarem em seus testes. Finalmente, todos os nodos de habilidade encontrados são listados, e baseado em seus pesos, um deles é selecionado aleatoriamente como habilidade a ser utilizada. Caso nenhum nodo de habilidade seja encontrado, a próxima árvore é então explorada. (Figura 4)

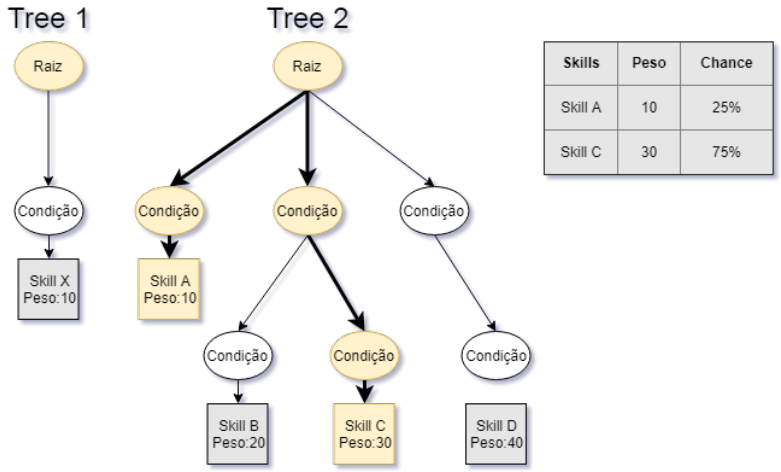


**Figura 4. Avaliação das árvores de decisão de um inimigo. Nodos dourados representam os nós explorados. A tabela no canto superior direito da imagem lista todos os nós de habilidade encontrados, com seus pesos e suas chances reais de serem selecionados. Nessa situação, caso a avaliação da primeira árvore tivesse encontrado a habilidade "*Skill X*", a segunda árvore não teria sido avaliada.**

## 5. Inteligência Artificial

O algoritmo *Monte Carlo Tree Search* (MCTS) [Chaslot et al. 2008] é tido como uma das formas mais eficientes e robustas de calcular jogadas contra oponentes em busca de um estado de vitória. Dentre suas vantagens, está o fato de que ele não depende de conhecimento externo do jogo, dependendo apenas de suas regras para realizar a simulação. Sua aplicação original é focada em jogos baseados em turnos onde cada jogador realiza uma ação de cada vez, como é o caso do jogo *GO* [Brügmann 1993]. Experimentos com aplicações do mesmo algoritmo em jogos de tempo real se provaram eficazes, apesar de apresentarem alguns problemas quando próximas do estado de vitória, fazendo com que o algoritmo penda para um estado de empate [Den Teuling and Winands 2011].

O comportamento original do MCTS é realizado da seguinte forma. Cada possível estado de jogo é representado por um nodo, e ligando esses nodos estão arestas, que representam as possíveis ações que podem ser tomadas dado o estado atual do jogo. Enquanto não é necessário tomar uma decisão, o agente simula diversos *playouts*. Cada *playout* representa uma expansão da árvore, onde, a partir de sua raiz, ações são simuladas até se encontrar um nodo-folha, quando mais ações não podem ser tomadas e se classifica como um estado de vitória ou derrota. Nodos criados e visitados guardam valores

de quantas vezes foram simulados e quantos estados de vitória encontraram (Figura 5). Como jogador e inimigo alternam turnos, cada nível da árvore representa o turno de um deles, e a quantidade de vitórias guardadas se refere apenas ao jogador que age no turno do nodo em questão. Quando o agente precisa tomar uma ação, ele consulta os nodos todos expandidos diretamente a partir da raiz e realiza a ação que tem maiores chances de vitória.

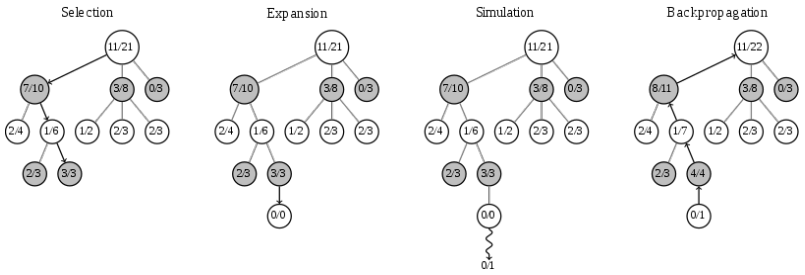


**Figura 5. Um *playout* do MCTS. Nodos escuros e claros representam turnos de diferentes jogadores, e cada aresta representa um movimento realizado. No valor dentro de cada nodo, o numerador representa a quantidade de estados de vitória encontrados para o jogador dono do turno, enquanto o denominador representa o total de simulações realizadas que passam por aquele nodo [Wikimedia Commons 2017].**

Estudos realizados por [Schaeffer et al. 2009] demonstram alguns dos problemas na aplicação do MCTS puro em jogos de turno onde todos os participantes tomam suas decisões em paralelo, usando como exemplo o clássico *Pedra, Papel e Tesoura*. Sua pesquisa sugere o uso de algoritmos para prever decisões realizadas pelo oponente baseado em jogadas anteriores e em suas chances de arrependimento. [Perick et al. 2012] expande essa pesquisa, analisando as consequências da aplicação do MCTS em jogos de turnos simultâneos e como a perda do fator determinístico pode ser combatida com diferentes formas de seleção de movimento. Uma das consequências da aplicação do MCTS a um jogo de turnos simultâneos é que se perde a diferença entre nodos de turno do jogador e do inimigo, fazendo assim com que todas as possíveis ações sejam na verdade o produto cartesiano das ações que o jogador e que o inimigo pode executar.

Paralelo a isso, [Lanctot et al. 2014] explora a possibilidade de otimizar a etapa de seleção de movimento combinando taxas de vitórias com valores heurísticos. Cada nodo, além de guardar quantas vitórias e visitas foram realizadas na expansão da árvore, passa também a guardar o maior resultado heurístico dentre todos os nodos-folha encontrados. Unindo taxa de vitórias com heurística, é feito um balanço baseado num valor $\alpha \in [0, 1]$ que interpola entre ambos valores.

## 6. Combinando DDA com IA

Para o sistema de batalha do *Capital* (detalhado na Sessão 4), vamos dividir a explicação em duas partes. Na sessão 6.1 detalharemos a coleta de informações por parte do DDA, logo em seguida explicando como o resultado do DDA pode ser usado para ajustar a dificuldade da IA que controla os inimigos do jogo na sessão 6.2.

## 6.1. Modelo de DDA

Um dos objetivos da criação de um modelo para gerenciar o DDA é trabalharmos com uma gama de variáveis capazes de dar ao game designer um controle fino em como o jogo gerencia sua dificuldade. Não entraremos no âmbito de quais são os valores ideais visto que eles podem sempre passar por mais iterações em busca do balanço ideal. Dito isso, tocaremos em algumas das diretrizes utilizadas no modelo de DDA do *Capital*.

Nosso sistema tem um componente cujo objetivo é dificultar o jogo além de suas regras bases . O nível de dificuldade em sua máxima amplitude é definido por $d \in [0, 1]$. $d = 1$ representa a máxima influência do sistema dificultador do jogo, enquanto $d = 0$ representa a total ausência de influência do mesmo. A fim de ter um maior controle na amplitude de dificuldade alcançada, definimos duas constantes $d_{min}$ e $d_{max}$ e introduzimos uma nova variável $d' \in [0, 1]$ usada para calcular o valor de $d$ como representado abaixo:

$$d = d' * (d_{max} - d_{min}) + d_{min} \tag{1}$$

Tendo $d'$ como nosso controle de dificuldade do jogo, passamos a definir que fatores iram influenciar no valor desse número. Para o *Capital*, definimos 2 fatores externos e 1 fator interno a cada batalha. Cada fator tem um valor modificador atrelado que, quando somados a uma dificuldade base, resultam na dificuldade do DDA. Fatores externos são definidos como influências que antecedem o início da batalha, se mantendo fixos ao longo de uma batalha. Fatores internos, por outro lado, reagem em tempo real a acontecimentos dentro da batalha, e por esse motivo são os mais delicados, devido ao risco elevado do jogador perceber o DDA em ação.

### 6.1.1. Histórico de batalhas

O primeiro fator externo que adicionamos ao modelo de DDA é o histórico de batalhas do jogador definido por $H = (h_0, h_1, ..., h_{n-1})$ onde $h_i \in [-1, 1]$. Retomando o sistema de pontuação de batalhas previamente explicado na sessão 4, ao sair vitorioso de uma batalha e receber sua recompensa, seu saldo final pode acabar em lucro ou prejuízo. Para fins de progressão do jogo, lucro é sempre preferível a prejuízo. Dessa forma, considerando que $b$ representa o saldo do jogador após pegar sua recompensa e $b_{initial}$ é o valor com o qual o jogador iniciou a batalha, adicionamos à $H$ o valor resultante a seguir:

$$max(-1, min(1, \frac{b_{player}}{b_{player_initial}} - 1)) \tag{2}$$

Tendo o histórico de resultados de batalhas, precisamos extrair dele um único valor $h_{mod}$ que será usado para o calculo da dificuldade final. Definimos aqui uma nova constante, denominada $h_{power}$, que define qual o peso de cada batalha para a soma final. A fim de aumentar a influência de batalhas mais recentes, realizamos a seguinte soma:

$$h_{mod} = \sum_{i=0}^{n-1} \left( h_i * \left( \frac{1}{n-i} \right)^{h_{power}} \right) \tag{3}$$

### 6.1.2. Saldo esperado

O segundo fator externo é calculado imediatamente ao iniciar uma batalha. Cada inimigo no jogo tem um saldo esperado do jogador $b_{expected}$, significando que o design do inimigo foi feito de forma que $b_{player} \approx b_{expected}$. O calculo realizado aqui é mais simples, se resumindo ao seguinte:

$$b_{mod} = \frac{b_{player} - b_{expected}}{b_{expected}} \tag{4}$$

### 6.1.3. Estado de batalha

O fator interno usado é um calculo heurístico em cima do estado atual da batalha. O propósito do fator interno é maior em facilitar uma batalha caso o jogador esteja tendo dificuldade do que dificultar caso contrário, portanto nós impedimos a dificuldade de ser incrementada no calculo realizado. Dado uma batalha qualquer num estado entre turnos, considerando que a função $W(b)$ representa quanto de seu saldo original um personagem ainda tem (Equação 5), o fator interno pode ser calculado como demonstrado na equação 6.

$$W(b) = max(0, min(1, \frac{b}{b_{initial}})) \tag{5}$$

$$w_{mod} = min(0, \frac{W(b_{player}) - W(b_{enemy})}{2}) \tag{6}$$

### 6.1.4. Calculo da dificuldade

Uma vez tendo o valor de cada um dos fatores calculados, precisamos atribuir pesos a cada um deles a fim de definir qual deve ser a influência de cada um na dificuldade. Dessa forma, definimos 3 novas constantes $h_{weight}$, $b_{weight}$ e $w_{weight}$ e finalmente temos o suficiente para calcular a dificuldade gerada pelo DDA, como demonstrado na fórmula a seguir:

$$d_h = h_{mod} * h_{weight}$$

$$d_b = b_{mod} * b_{weight}$$

$$d_w = w_{mod} * w_{weight}$$

$$d' = max(0, min(1, 0.5 + d_h + d_b + d_w)) \tag{7}$$

### 6.2. Ajustando dificuldade da IA

Para garantir que possamos aplicar uma gama alta de níveis de dificuldade, precisamos garantir que nossa IA seja capaz de jogar da melhor forma possível. Para esse fim, aplicamos o MCTS com algumas das adaptações apresentadas na Sessão 5.

Similar a solução implementada por [Perick et al. 2012], nós fazemos a expansão dos nodos baseado no produto cartesiano dos possíveis movimentos do jogador e do inimigo. Dessa forma, caso o inimigo tenha as habilidades A e B, e o jogador tenha C e D, os nodos que criamos são $\{\{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}\}$, assim cobrindo todas as possibilidades de resultados daquele turno. Um ponto interessante aqui é que a IA só leva em consideração habilidades já utilizadas pelo jogador, criando assim a possibilidade de um ataque surpresa por parte de alguém que esteja jogando contra a IA.

Durante a abertura de nodos, aplicamos a mesma estratégia de [Lanctot et al. 2014], na qual balanceamos quantidades de estados de vitória $v$ com uma avaliação heurística acumulativa dos nodos da árvore. Dada uma função heurística definida pela Equação 8, usamos uma constante $\alpha \in [0, 1]$ para definir o peso de cada uma das duas medidas. O resultado final da avaliação de um nodo é definido por $m$

$$w = 0.5 + \frac{W(b_{player}) - W(b_{enemy})}{2} \tag{8}$$

$$m = v * \alpha + w * (1 - \alpha) \tag{9}$$

Finalizado a etapa de expansão da árvore e entrando na seleção de nodo, não temos mais como prever com exatidão qual vai ser o próximo estado do jogo visto que não temos como saber se o adversário usará a habilidade A ou B. Similar ao funcionamento do Minimax, a estratégia é prever qual o melhor movimento que o adversário pode fazer, e trabalhar em cima dele. Agrupamos os nodos filhos da expansão e calculamos uma média baseada em cada habilidade, então usando a habilidade de pontuação mais alta como habilidade referencia do adversário. Eliminamos da seleção todos nodos onde o adversário não usa a habilidade em questão.

Tendo agora um nodo para cada habilidade que a ia pode selecionar, o caminho natural seria selecionar o de maior probabilidade de vitória. Nosso objetivo, porém, é integrar isso com o $d$ gerado pelo DDA, visando interpolar níveis de dificuldade de forma convincente. No design das árvores de decisão de cada inimigo demonstrada na sessão 4, cada habilidade tem um peso atrelado, o qual vamos chamar de $g$. A ideia é que, $g$ define um comportamento sub-ótimo, enquanto $m$ define um comportamento ótimo. A utilização da dificuldade dinâmica está em interpolar ambos valores. Definimos $g'$ e $m'$ como versões normalizadas desses pesos, e então calculamos o peso final $p$ de cada nodo. A fim de fazer com que o peso de $m$ dê preferência a habilidades de maior chance, definimos uma última constante $m_{power}$.

$$g'_i = \frac{g_i}{\sum_j^n g_j} \tag{10}$$

$$m'_i = \frac{m_i^{m_{power}}}{\sum_j^n m_j^{m_{power}}} \tag{11}$$

$$p_i = m'_i * d + g'_i * (1 - d) \tag{12}$$

Tendo os pesos $p_i$ de cada nodo, tudo que nos resta é selecionar um deles aleatoriamente baseado em seus pesos

## 7. Resultados e Limitações

A implementação descrita ao longo desse artigo apresentou resultados satisfatórios. A estratégia aplicada na manipulação da dificuldade se mantém discreta frente aos olhos do jogador por se esconder atrás do fator aleatório das batalhas. A quantidade de constantes configuráveis permitem um controle preciso no controle do DDA por parte de um Game Designer, possibilitando respostas rápidas na manipulação da dificuldade no caso do jogador entre numa sequência de resultados ruins

A aplicação do DDA e IA nas batalhas não apenas permite o controle na dificuldade, mas também facilita o design das árvores de decisão de cada padrão de inimigo. No momento em que uma IA pode tomar a decisão de quando é melhor realizar algum comportamento específico, podemos simplificar as árvores, delegando algumas responsabilidades para a IA e assim tornando o design mais orgânico.

A implementação de nossa própria variação do MCTS no jogo também se mostrou muito eficaz. Simulações de batalhas entre IAs com níveis de dificuldade diferentes obtiveram resultados esperados, onde mesmo com o fator aleatório da batalha influenciando, IAs com dificuldade mais alta obtiveram resultador proporcionais.

Dentre as limitações, se identificaram comportamentos estranhos em estados nos quais a IA não encontra forma alguma de vencer a batalha. Seu comportamento acaba se assemelhando muito ao de $d = 0$, onde a IA não tem formas de prever a eficácia de seus movimentos e acaba aparentando que ela não está se esforçando para ganhar a batalha, mesmo numa situação onde $d = 1$.

## 8. Conclusões

Ao longo desse artigo, exploramos o conceito de *Flow*, explorando o papel da dificuldade em uma boa experiência de jogo, as vantagens e importância de um sistema de DDA, e sua aplicação em um sistema de combate simultâneo em turnos. Detalhamos como desenhar uma IA que funciona em conjunto com o design dos inimigos e como controlar seu nível de dificuldade de forma dinâmica.

Para trabalhos futuros se considera investigar mais a fundo o impacto do DDA num sistema de batalha de turnos, explorando outras formas de dificultar o jogo, como manipulando danos de habilidades, recompensas com bônus, possibilidades de esquiva ou gerenciamento de recursos disponibilizados ao jogador, esse último sendo similar ao pesquisado por [Hunicke 2005]. Testes poderiam ser executados em grupos, avaliando eficacia da manipulação da dificuldade e percepção das mudanças na mesma.

## Referências


Andrade, G., Ramalho, G., Gomes, A. S., and Corruble, V. (2006). Dynamic game balancing: An evaluation of user satisfaction. *AIIDE*, 6:3–8.

Bento, D., Lewandowski, E., Felix, F., Marshall, G., and Costa, V. (2018). Capital: Sin & money. `https://beastarman.itch.io/capital`.

Brügmann, B. (1993). Monte carlo go. Technical report, Citeseer.

Cechanowicz, J. E., Gutwin, C., Bateman, S., Mandryk, R., and Stavness, I. (2014). Improving player balancing in racing games. In *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play*, pages 47–56. ACM.

Chanel, G., Rebetez, C., Bétrancourt, M., and Pun, T. (2008). Boredom, engagement and anxiety as indicators for adaptation to difficulty in games. In *Proceedings of the 12th international conference on Entertainment and media in the ubiquitous era*, pages 13–17. ACM.

Chaslot, G., Bakkes, S., Szita, I., and Spronck, P. (2008). Monte-carlo tree search: A new framework for game ai. In *AIIDE*.

Cruz, C. A. and Uresti, J. A. R. (2017). Player-centered game ai from a flow perspective: Towards a better understanding of past trends and future directions. *Entertainment Computing*, 20:11–24.

Csikszentmihalyi, M. (1990). Flow. the psychology of optimal experience. new york (harperperennial) 1990.

Den Teuling, N. G. and Winands, M. H. (2011). Monte-carlo tree search for the simultaneous move game tron. *Univ. Maastricht, Netherlands, Tech. Rep*.

Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 429–433. ACM.

Jennings-Teats, M., Smith, G., and Wardrip-Fruin, N. (2010). Polymorph: dynamic difficulty adjustment through level generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 11. ACM.

Lanctot, M., Winands, M. H., Pepels, T., and Sturtevant, N. R. (2014). Monte carlo tree search with heuristic evaluations using implicit minimax backups. *arXiv preprint arXiv:1406.0486*.

Nakamura, J. and Csikszentmihalyi, M. (2014). The concept of flow. In *Flow and the foundations of positive psychology*, pages 239–263. Springer.

Perick, P., St-Pierre, D. L., Maes, F., and Ernst, D. (2012). Comparison of different selection strategies in monte-carlo tree search for the game of tron. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 242–249. IEEE.

Salen, K., Tekinbaş, K. S., and Zimmerman, E. (2004). *Rules of play: Game design fundamentals*. MIT press.

Schaeffer, M. S. N. S. J., Shafiei, N., et al. (2009). Comparing uct versus cfr in simultaneous games.

Smeddinck, J. D., Mandryk, R. L., Birk, M. V., Gerling, K. M., Barsilowski, D., and Malaka, R. (2016). How to present game difficulty choices?: Exploring the impact on player experience. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5595–5607. ACM.

Sutoyo, R., Winata, D., Oliviani, K., and Supriyadi, D. M. (2015). Dynamic difficulty adjustment in tower defence. *Procedia Computer Science*, 59:435–444.

Wikimedia Commons (2017). The 4 steps of a monte carlo search tree expansion. File: MCTS (English) - Updated 2017-11-19.svg.