

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
ESCOLA DE ENGENHARIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ROGIEL JOSIAS SULZBACH

# **Método de classificação de build orders em StarCraft II**

Porto Alegre

2016

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
ESCOLA DE ENGENHARIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ROGIEL JOSIAS SULZBACH

## **Método de classificação de build orders em StarCraft II**

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para Graduação em Engenharia Elétrica

Orientador: Prof. Dr. Altamiro Amadeu Susin

Porto Alegre

2016

ROGIEL JOSIAS SULZBACH

## Método de classificação de build orders em StarCraft II

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para Graduação em Engenharia Elétrica

---

**Prof. Dr. Altamiro Amadeu Susin**  
Orientador - UFRGS

---

**Prof. Dr. Ály Ferreira Flores Filho**  
Chefe do Departamento de Engenharia Elétrica (DELET) - UFRGS

Aprovado em ??? de dezembro de 2016.

data

BANCA EXAMINADORA

---

**Prof. Dr. Luiz Fernando Ferreira**  
UFRGS

---

**Prof. Dr. Marcelo Soares Lubaszewski**  
UFRGS

---

**Prof. Dr. Tiago Roberto Balen**  
UFRGS

# Resumo

RESUMO  

Palavras-chave: ...  

# Abstract

ABSTRACT   abstract

Keywords: ...   keywords

# Lista de Figuras

Figura 1 – Exemplo de uma distribuição de probabilidade de uma ação com 2 repetições . . . . .	12
Figura 2 – Exemplo da distribuição de probabilidade da ação $A_1$ . A curva em azul representa a primeira repetição da ação (nos tempos $t_1$ e $t_2$ ), a curva em laranja representa a segunda repetição (nos tempos $t_3$ e $t_4$ ) e a curva em verde representa a terceira repetição da ação (tempos $t_6$ e $t_7$ ). . . .	19
Figura 3 – Exemplo da distribuição de probabilidade da ação $A_1$ . Como a ação $A_2$ somente é executada uma única vez no exemplo, apenas uma repetição é mostrada no gráfico. A ação corresponde aos instantes de tempo $t_5$ e $t_8$ . . . .	20
Figura 4 – Forma gráfica de distribuição estatística de uma ação <i>Gateway</i> e suas repetições . . . . .	28
Figura 5 – Forma gráfica de distribuição estatística de uma ação <i>Adept</i> e suas repetições . . . . .	29

# Lista de Tabelas

Tabela 1	– Um exemplo de uma <i>build order</i> com as ações $A_1$ e $A_2$ ocorrendo nos tempos $t_n$ . Observa-se nas últimas 2 linhas da tabela que há uma inversão de sequência de execução das ações. Esta alteração não implica que as <i>build order</i> sejam diferentes, é muito provável que os tempos $t_5$ , $t_6$ , $t_7$ e $t_8$ sejam muitos próximos e são independentes. . . . .	17
Tabela 2	– Resultado da autoclassificação . . . . .	29
Tabela 3	– Resultado da validação cruzada . . . . .	30
Tabela 4	– Resultado da validação cruzada . . . . .	31

# Lista de Abreviaturas e Siglas

BO	<i>Build Order</i>
SC2	<i>StarCraft II</i>
PDF	<i>Probability Distribution Function</i>
CDF	<i>Cumulative Distribution Function</i>
RTS	<i>Real Time Strategy</i>



# Sumário

1	INTRODUÇÃO . . . . .	9
1.1	Sobre o StarCraft II . . . . .	9
1.2	O uso de aprendizado de máquina em jogos de estratégia em tempo real . . . . .	10
2	REVISÃO BIBLIOGRÁFICA . . . . .	11
2.1	Trabalhos anteriores . . . . .	11
2.2	A estrutura estatística de uma <i>build order</i> . . . . .	12
2.3	Fator de <i>branching</i> de um jogo de StarCraft: BroodWar . . . . .	13
2.4	O Teorema de Bayes . . . . .	13
3	MÉTODO . . . . .	15
3.1	Extração dos dados . . . . .	15
3.2	<i>Dataset</i> de referência . . . . .	16
3.3	Treinamento . . . . .	17
3.3.1	Forma Matricial . . . . .	20
3.4	Classificação . . . . .	22
3.4.1	Forma Matricial . . . . .	22
3.5	Implementação . . . . .	22
4	RESULTADOS E DISCUSSÕES . . . . .	27
4.1	Autoclassificação . . . . .	29
4.2	Validação cruzada . . . . .	30
4.3	Teste de ruído . . . . .	30
5	CONCLUSÕES . . . . .	32
6	PROPOSTAS DE TRABALHOS FUTUROS . . . . .	33
	REFERÊNCIAS BIBLIOGRÁFICAS . . . . .	34
	Glossário . . . . .	35
	ANEXO A – LISTA DE NOMES DE AÇÕES UTILIZADAS . . . . .	38

# 1 Introdução

## 1.1 Sobre o StarCraft II

StarCraft II é um jogo de estratégia militar em tempo-real desenvolvido pela *Blizzard Entertainment* onde três facções disputam partidas *multiplayer* (online com múltiplos jogadores) em modalidade 1 contra 1. O objetivo do jogo consiste em conseguir destruir todas as estruturas do adversário que provém infraestrutura para a construção e manutenção do exército.

O jogo possui um sistema de economia, onde para que um jogador possa treinar ou desenvolver uma tecnologia, é necessário que primeiro sejam extraído recursos do ambiente virtual de jogo. O jogo também apresenta um sistema de "árvore tecnológica" onde há um encadeamento de pre-requisitos para o treinamento e construção de unidades de exército mais avançadas. É a existência desta árvore tecnológica que possibilita que seja possível inferir e prever uma determinada estratégia do jogador.

O modo mais comum de execução de uma estratégia é utilização de *build orders*. Uma *build order* é uma sequência de ações tomadas por um jogador no decorrer do jogo, que, quando executadas de forma correta, oferecem alguma vantagem estratégica para o jogador. Muitas build orders são padronizadas e otimizadas por jogadores profissionais durante o treino e são popularizadas em campeonatos mundiais.

Neste trabalho será desenvolvido e implementado um método de classificação para *build order* de partidas de jogadores profissionais de StarCraft II. Foi feita a escolha de utilizar partidas profissionais pois são jogadores com elevado conhecimento do jogo e reagem de forma ótima para várias situações inusitadas ou inesperadas, o que reduz a variabilidade das medidas extraídas dos arquivos de *replay* do jogo. Para a extração de dados foi utilizado um pacote de *replays* (arquivo binário que codifica todas as ações tomadas em um jogo) de competições profissionais de StarCraft II durante o ano de 2016.

Uma aplicação direta para o método proposto é o desenvolvimento de uma inteligência artificial que seja capaz de tomar decisões ao longo de um jogo de forma verossímil à um jogador humano, conforme proposto em (SYNNAEVE; BESSIERE, 2011a).

## 1.2 O uso de aprendizado de máquina em jogos de estratégia em tempo real

O uso de aprendizado de máquina em jogos de estratégia em tempo real (RTS) pode ser dividido em vários problemas: táticas, estratégias e *micro management* conforme definido por (SYNNAEVE; BESSIERE, 2011b).

Táticas se refere ao posicionamento de unidades no mapa e é diretamente dependente da forma com a qual as unidades interagem entre si no jogo. Dessa forma, uma solução para este problema deve considerar o mapa e as interações entre as mecânicas de cada unidade.

Os problemas de estratégia se referem ao plano geral de jogo. A estratégia surge de uma expectativa para o desenvolver do jogo, por exemplo, supondo que um jogador deseja investir fortemente na sua economia para que consiga construir unidades tecnológicas de maior custo. A estratégia é a forma com a qual ele vai conseguir atingir este objetivo. Em geral, a estratégia é independente do mapa em que jogo está sendo jogado, mas é diretamente relacionado à raça do oponente, suas escolhas tecnológicas e sua gerência da economia (geralmente denominado de *macro management*). A estratégia de um jogador está diretamente relacionada com a sua escolha *build order*.

Por fim, os problemas de *micro management* consistem na gerência de unidades individuais, ou seja, o jogador faz o controle de cada unidade do exercito de forma individual. O *micro management* é uma especialização das táticas, mas se refere à cada unidade individual ao invés do exército inteiro.

## 2 Revisão Bibliográfica

### 2.1 Trabalhos anteriores

"A Data Mining Approach to Strategy Prediction"(WEBER; MATEAS, 2009) apresentaram uma forma de expressar uma *build order* de StarCraft Brood War em um vetor de *features* para processamento bem como a performance de quatro algoritmos de classificação. No trabalho concluíram que os diferentes algoritmos possuíam performance diferenciada nos diversos estágios de jogo. O modelo de codificação de features proposto baseava-se num critério de primeira-aparição, isto é, o vetor de *features* contém o instante em que a primeira instância de uma dada unidade ou estrutura ocorria no jogo e, caso não houvesse ocorrência, utilizava um valor padronizado de zero.

Adicionalmente, a fim de simular efeitos de *scouting* ruído foi adicionado nos vetores de *features* como forma de gerar variação nos *timings* de cada ação e então analisar a performance dos algoritmos. Foi concluído que todos os algoritmos perdiam precisão proporcional à quantidade de ruído adicionada, no entanto o algoritmo de k-NN (*k-nearest neighbors*) não degradou a performance de forma tão drástica quanto os outros algoritmos. Também foi adicionado outro teste de ruído para simular informação incompleta: lacunas foram inseridas no vetor de *features* a fim de emular a situação em que o jogador não consegue extrair uma informação do jogo do oponente. Com este tipo de ruído, a conclusão foi de que a precisão dos algoritmos de decaía de forma linear com o nível de ruído adicionado no vetor.

Em "A Bayesian Model for Opening Prediction in RTS Games with Application to StarCraft"(SYNNAEVE; BESSIERE, 2011a), baseado no trabalho anterior de (WEBER; MATEAS, 2009), realizaram o desenvolvimento de um modelo Bayesiano para classificação de build orders de StarCraft: Brood War utilizando um método de extração de *features* de ações significativas dos replays. A extração do replay considerava estruturas na árvore de tecnologia e considerava apenas a primeira construção de uma estrutura e o índice da sequência de construção era utilizado para treinar o sistema. O método era capaz de identificar a abertura do jogador após a construção de, na média, 10 estruturas. Contudo, devido ao número limitado de *features*, era possível enumerar todas as possibilidades de jogos possíveis (em torno de 1000 por raça). O método era adequado para classificação em que a informação era limitada, por exemplo, era adequada para o contexto de uma inteligência artificial de toma suas decisões baseada na sua capacidade de *scouting* (tentar descobrir a estratégia de um outro jogador utilizando uma unidade que descobre cada estrutura feita pelo oponente).

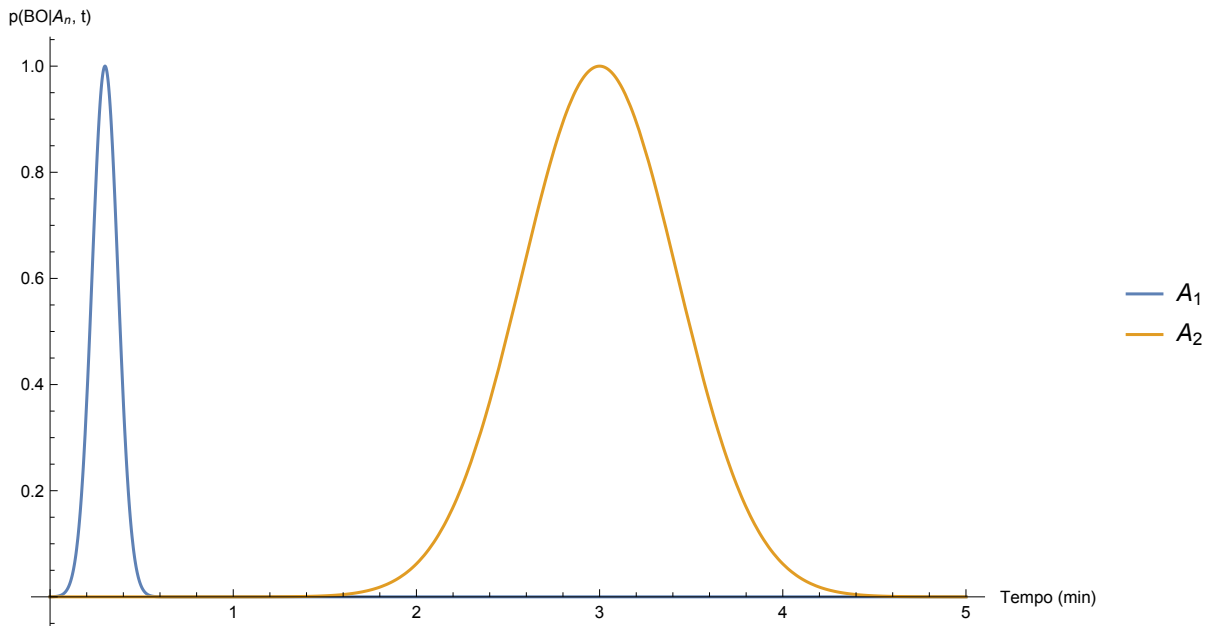
(SYNNAEVE; BESSIERE, 2011b) apresenta uma análise extensiva para técnicas e dificuldades encontradas ao aplicar algoritmos de aprendizado de máquina em diversos tipos de jogo, incluindo jogos de estratégia em tempo real (RTS).

## 2.2 A estrutura estatística de uma *build order*

Uma build order genérica pode ser expressa como uma sequência de ações cujo valor tempo de execução médio e desvio padrão estão relacionados com uma função distribuição de probabilidade.

Seja  $A$  uma ação arbitrária,  $A_n$  a sequência da ação durante o jogo (primeira vez que ela é executada, segunda, terceira, etc...),  $BO$  seja uma *build order* em que a sequência de ações  $A_n$  foram executadas e  $t$  o tempo de jogo em minutos. Dessa forma, podemos expressar a distribuição de probabilidade de uma sequência da forma indicada na Figura 1:

Figura 1 – Exemplo de uma distribuição de probabilidade de uma ação com 2 repetições



Na Figura 1 é possível observar que a primeira vez que a ação é executada, a tolerância para atraso é pequena, contudo, na segunda execução a tolerância de atraso é muito maior, pois o efeito da árvore tecnológica implica que uma ação atrasada irá atrasar as duas ações dependentes.

A tolerância de execução se faz necessária pois um jogador pode atrasar alguns segundos por múltiplos motivos durante o jogo, embora ele ainda esteja executando a mesma build-order.

Para uma *build order* real, haverá uma distribuição de probabilidade para cada uma das ações que podem oscilar em torno de 30 a 40 dependendo da facção do jogador.

## 2.3 Fator de *branching* de um jogo de StarCraft: BroodWar

Em teoria dos jogos, *branching factor* é o número de nós-filho em cada um dos nós de ações possíveis no jogo, isto é, é o número de opções que um jogador tem disponível em cada momento. referência

Infelizmente não há análises científicas feitas para a complexidade de um jogo de StarCraft II, mas é possível realizar uma comparação utilizando o jogo antecessor da série, StarCraft: Brood War lançado em 1998. De acordo com (WEBER; MATEAS, 2009), StarCraft: BroodWar possui um fator de branching estimado médio de  $1 \times 10^6$  que, comparado à um jogo de xadrez onde a média é de 35, é um valor extremamente alto. O alto nível de complexidade força que seja utilizado um modelo probabilístico ao invés de uma análise determinista, uma vez que é impossível obter, ou sequer gerar, uma amostra para todas as possibilidades de jogos. Devido a novas mecânicas de jogo introduzidas em StarCraft II espera-se a complexidade seja superior à de seu antecessor.

## 2.4 O Teorema de Bayes

O Teorema de Bayes relaciona a probabilidade de um evento associado à uma restrição. A definição matemática do teorema é dada pela Equação 2.1:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (2.1)$$

onde  $A$  é um evento,  $B$  é a restrição do evento  $A$ . Dessa forma, é possível determinar a probabilidade do evento  $A$  ter acontecido, dado que o evento  $B$  foi observado.

Em termos da análise de *build orders*, pode-se aplicar o teorema conforme a Equação 2.2:

$$P(BO|A, t) = \frac{P(A|BO, t)P(BO)}{P(A, t)} \quad (2.2)$$

onde  $BO$  é uma *build order* qualquer que deseja-se estimar a probabilidade de estar sendo executada, dado que o jogador executou a ação  $A$  no tempo  $t$ . Para tanto, precisa-se determinar 3 fatores:

- $P(A, t)$ : a probabilidade de um jogador executar a ação  $A$  no instante  $t$ , independe da *build order*;

- $P(BO)$ : a probabilidade de um jogador executar a *build order*  $BO$ , independente da ação que o jogador está executando;
- $P(A|BO, t)$ : a probabilidade de ter executado a ação  $A$  supondo que o jogador está executando a *build order*  $BO$  no instante  $t$ ;

## 3 Método

Neste trabalho foi utilizado Python 2.7 como a linguagem de programação para a implementação do método. Este motivo foi guiado principalmente pelo fato da biblioteca oficial de processamento de replays oferecida pela Blizzard Entertainment, Inc. (desenvolvedor do StarCraft II) ser em Python. A biblioteca de processamento de replays oficial, *s2protocol*(Blizzard Entertainment, Inc., 2013) é de código-fonte aberto e está disponível no GitHub. Demais códigos utilizados são parte de uma distribuição padrão do Python 2.7 ou foram desenvolvidos para o trabalho.

### 3.1 Extração dos dados

Os replays de StarCraft II são arquivos do tipo MoPAQ, um formato proprietário desenvolvido pela Blizzard Entertainment nos anos 90 para uso em seus jogos. Embora o formato seja proprietário é possível localizar na internet uma gama de implementações ou documentação desenvolvida por engenharia reversa. Um arquivo MPQ é análogo à um arquivo ZIP e contém um índice de arquivos e blocos de conteúdo. Nos *replays* há uma série de *stream* de eventos de jogo e neste trabalho somente será utilizado o stream chamado "Tracker Events" que são eventos cujo conteúdo é destinado para ferramentas que desejam processar os replays, ao contrário de informações úteis para a simulação do jogo. Este *stream* está armazenado como "replay.tracker.events" dentro do MPQ.

O *stream* de *Tracker Events* foi processado utilizando uma classe em Python que extrai as seguintes informações do evento:

- Nome (tipo) do evento;
- *Game loop* do evento (o número de iterações do loop principal do jogo);
- ID do jogador que gerou o evento;
- Estrutura de dados específica de cada evento;

A expressão de conversão de *game loops* para o tempo, em segundos é dada pela Equação 3.1.

$$t_{seg} = 16 \cdot n_{loops} \cdot \frac{26}{36} \cdot \frac{131}{148} \quad (3.1)$$

onde  $t_{seg}$  é o tempo em segundos,  $n_{loops}$  é o número de *game loops*.



Dentre os vários tipos de eventos contidos neste stream, 3 destes eventos são interessantes na análise:

**NNet.Replay.Tracker.SUnitBornEvent** evento disparado para cada unidade/estrutura criada no jogo. Este evento somente é despachado para unidades que aparecem no campo de batalha de forma "pronta", isto é, no momento em que a unidade pode ser vista pelo jogador, ela já pode ser controlada.

**NNet.Replay.Tracker.SUnitInitEvent** semelhante ao evento SUnitBornEvent, mas este evento é despachado para unidades que são construídas diretamente no campo de batalha e não estão disponíveis para jogo imediatamente no instante do evento. Embora a unidade/estrutura neste evento não seja imediatamente utilizável, no método proposto, apenas o tempo em que usuário inicia a construção é considerado.

**NNet.Replay.Tracker.SUpgradeEvent** evento disparado para cada melhoramento de exército realizado pelo jogador. Estes eventos são muito importantes pois podem indicar ou refinar a intenção do jogador.

Apenas uma informação é extraída da estrutura de dados específica dos eventos: o nome da unidade, estrutura ou melhoramento feito pelo jogador. Este nome é único para cada tipo de unidade, estrutura ou melhoramento e cada um desses nomes são considerados como ações individuais que podem ser executadas pelo jogador. A única exceção é a exclusão de unidades do tipo trabalhadoras e estruturas que oferecem suprimento ao jogador. Estas unidades foram removidas porque num jogo regular estas ações ocorrem muitas vezes e há muita variância na execução delas, logo, incorrendo num prejuízo na qualidade do resultado final.

Uma vez que após os 6 minutos de jogo, a partida se torna reativa, ao invés de algorítmica, a sequência de eventos é truncada até os 6 minutos de jogo. Isto garante que as informações tenham menor variabilidade. Ao decorrer da partida, as ações executadas por um jogador começam a se tornar em resposta das ações do outro jogador e, portanto, não há sentido realizar uma classificação de *build order* para ações que ocorrem além de um certo limite de tempo.

## 3.2 *Dataset* de referência

Como não há nenhum índice de replays e build orders disponível publicamente o conjunto de replays utilizado para treinamento e validação foi classificado manualmente utilizando replays de diversos campeonatos do ano de 2016

**Intel Extreme Masters 10** : 19 *replays*

**Intel Extreme Masters 11** : 35 *replays*

**WCS Spring Championship 2016** : 54 *replays*

totalizando 108 *replays*.

Para a classificação manual das estratégias cada replay foi assistido no jogo e classificado conforme as regras abaixo:

- A composição de exército de um jogador no instante do primeiro ataque realizado por ele;
- Se o jogador fosse atacado por outro e tivesse mais de 8 unidades perdidas, o *replay* era descartado;
- Caso não houvesse investida por parte dos dois jogadores até os 6 minutos de jogo, a composição de exército do jogador aos 6 minutos de jogo era classificada.

Um exército em StarCraft II pode ser composto por mais de 10 tipos de unidades diferentes, contudo isto é incomum. De forma geral, exércitos são compostos por um grande número de unidades básicas de ataque e algumas unidades de suporte. Na classificação, as unidades de suporte foram desprezadas na composição do exército. Esta regra é violada somente caso a unidade de suporte seja incomum ou trouxe um benefício significativo para o jogador, como estratégias do tipo *rush* (criar um exército o mais rápido possível, geralmente em sacrifício da economia).

### 3.3 Treinamento

O treinamento foi realizado de forma simples: a sequência de ações para cada *replay* era iterada e o tempo de execução da ação era gravado em uma lista. Uma lista separada era usada para cada repetição. Por exemplo, supondo que as *build orders*  $BO_1$  e  $BO_2$  sejam duas amostras com o mesmo *label*:

Tabela 1 – Um exemplo de uma *build order* com as ações  $A_1$  e  $A_2$  ocorrendo nos tempos  $t_n$ . Observa-se nas últimas 2 linhas da tabela que há uma inversão de sequência de execução das ações. Esta alteração não implica que as *build order* sejam diferentes, é muito provável que os tempos  $t_5$ ,  $t_6$ ,  $t_7$  e  $t_8$  sejam muito próximos e são independentes.

$BO_1$		$BO_2$	
<b>T</b>	<b>A</b>	<b>T</b>	<b>A</b>
$t_1$	$A_1$	$t_2$	$A_1$
$t_3$	$A_1$	$t_4$	$A_1$
$t_5$	$A_2$	$t_6$	$A_1$
$t_7$	$A_1$	$t_8$	$A_2$

Dessa forma, as duas listas terão o seguinte conteúdo:

$$T_{A_1} = \{\{t_1, t_2\}, \{t_3, t_4\}, \{t_6, t_7\}\} \quad (3.2)$$

onde  $T_{A_1}$  é o vetor de tempos da execução de cada repetição da ação  $A_1$  nos múltiplos *replays* processados.  $\{t_1, t_2\}$  representa os tempos da primeira repetição,  $\{t_3, t_4\}$  da segunda e  $\{t_6, t_7\}$  da terceira.

$$T_{A_2} = \{\{t_5, t_8\}\} \quad (3.3)$$

onde  $T_{A_2}$  é o vetor de tempos da execução de cada repetição da ação  $A_2$  nos múltiplos *replays* processados. Esta ação possui uma única repetição.

Seja  $\mu(T_x, R)$  a média do vetor de tempo  $T_x$  para a repetição  $R$  e  $\sigma(T_x, R)$  o desvio padrão do vetor de tempo  $T_x$  para a repetição  $R$ , então a distribuição  $p$  de probabilidade de uma ação pode ser definida como:

$$p_{T_x}(t, R) = Ae^{-\frac{(t-\mu(T_x, R))^2}{\sigma(T_x, R)^2}} \quad (3.4)$$

onde  $A$  é a frequência com que um par ação-repetição ocorre dentre todas as amostras de treinamento utilizadas,  $\mu$  é o tempo médio de execução do par ação-repetição e  $\sigma$  é o correspondente desvio-padrão.

A Equação 3.4 apresenta a função distribuição de probabilidade para uma ação. Isto é, representa a probabilidade de uma ação qualquer  $A$  pertencer a uma *build order*  $BO_1$  dado que a ação foi executada pelo jogador no instante  $t$ .

Esta função é uma variação da função distribuição de probabilidade Gaussiana, no entanto, é ajustada de forma que, para o ponto médio, seu valor seja unitário (para  $A = 1$ ) ou tenha o valor de  $A$ , que indica a frequência com que uma ação ocorreu nos *datasets* de treinamento.

Para o cálculo da média ( $\mu$ ) e desvio padrão ( $\sigma$ ) os itens internos do vetor  $T_x$  são utilizados. Para o exemplo apresentado nas Equações 3.2 e 3.3, as médias são dadas pelos vetores das Equações 3.5 e 3.6, respectivamente:

$$\mu_{A_1} = \left\{ \frac{t_1 + t_2}{2}, \frac{t_3 + t_4}{2}, \frac{t_6 + t_7}{2} \right\} \quad (3.5)$$

$$\mu_{A_2} = \left\{ \frac{t_5 + t_8}{2} \right\} \quad (3.6)$$

O cálculo de desvio padrão para os vetores das Equações 3.2 e 3.3 foi omitido no exemplo pois a expressão é complexa e não influencia no entendimento e um valor genérico é apresentado nas Equações 3.7 e 3.8.

$$\sigma_{A_1} = \{\sigma_{A_1,1}, \sigma_{A_1,2}, \sigma_{A_1,3}\} \quad (3.7)$$

$$\sigma_{A_1} = \{\sigma_{A_2,1}\} \quad (3.8)$$

Neste exemplo, de forma a simplificar o cálculo, assume-se que a frequência de cada par ação-repetição é unitária.

É possível substituir os valores de média e desvio-padrão das Equações 3.5, 3.6, 3.7 e 3.8 na função distribuição de probabilidade da Equação 3.4. O vetor final é denominado de "vetor de probabilidades".

Com os resultados de média (Equações 3.5 e 3.6) e desvio padrão (Equações 3.7 e 3.8), é possível obter um gráfico do formato da distribuição de probabilidade conforme representado nas Figuras 2 e 3.

Figura 2 – Exemplo da distribuição de probabilidade da ação  $A_1$ . A curva em azul representa a primeira repetição da ação (nos tempos  $t_1$  e  $t_2$ ), a curva em laranja representa a segunda repetição (nos tempos  $t_3$  e  $t_4$ ) e a curva em verde representa a terceira repetição da ação (tempos  $t_6$  e  $t_7$ ).

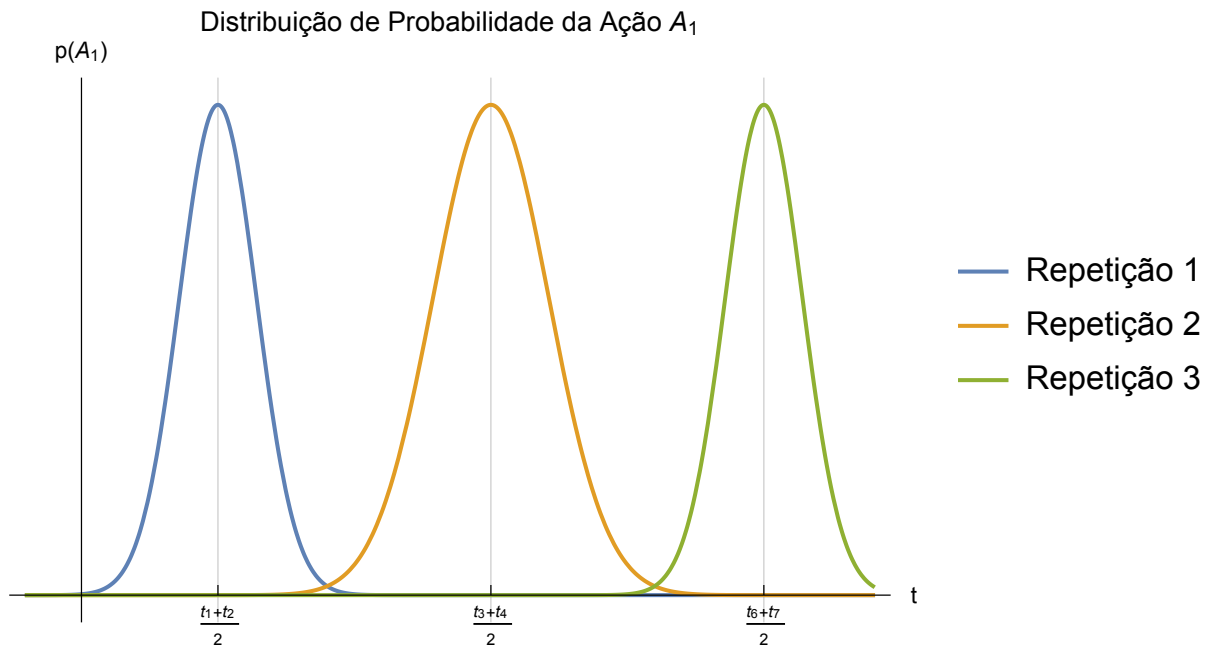
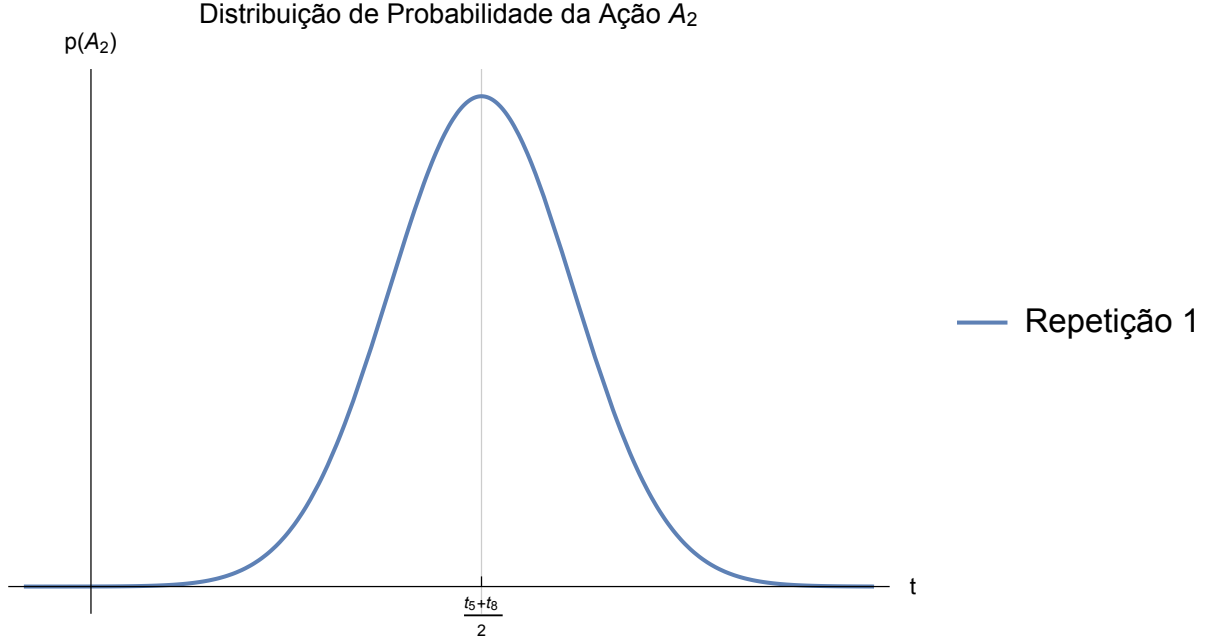


Figura 3 – Exemplo da distribuição de probabilidade da ação  $A_1$ . Como a ação  $A_2$  somente é executada uma única vez no exemplo, apenas uma repetição é mostrada no gráfico. A ação corresponde aos instantes de tempo  $t_5$  e  $t_8$ .



### 3.3.1 Forma Matricial

O treinamento pode ser realizado de forma matricial. Onde as colunas indicam as ações e as linhas indicam as repetições de cada ação respectiva. Dessa forma, é possível definir uma matriz de médias e desvios padrões.

Dado um conjunto de  $N$  ações que podem se repetir até  $M$  vezes, então, é possível definir uma matriz  $M \times N$ , cujos elementos são formados pelo tempo médio de execução de cada par ação-repetição, denominada de "matriz de média do tempo de execução", expressa na Equação 3.9.

$$\mu = \begin{pmatrix} \mu_{A_1,1} & \mu_{A_2,1} & \cdots & \mu_{A_N,1} \\ \mu_{A_1,2} & \mu_{A_2,2} & \cdots & \mu_{A_N,2} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{A_1,M} & \mu_{A_2,M} & \cdots & \mu_{A_N,M} \end{pmatrix} \quad (3.9)$$

De forma análoga a matriz de média do tempo de execução, é possível definir a "matriz de desvio padrão do tempo de execução" para os valores de desvio padrão do tempo de execução de cada par ação-repetição e a outra matriz denominada "matriz de frequência de ocorrência de repetição" que define a frequência de ocorrência de um dado par ação-repetição no conjunto de amostras observadas. A definição formal da matriz de

desvio padrão do tempo de execução está apresentada na Equação 3.10. A definição formal da matriz de frequência de repetição está apresentada na Equação 3.11.

$$\sigma = \begin{pmatrix} \sigma_{A_1,1} & \sigma_{A_2,1} & \cdots & \sigma_{A_N,1} \\ \sigma_{A_1,2} & \sigma_{A_2,2} & \cdots & \sigma_{A_N,2} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{A_1,M} & \sigma_{A_2,M} & \cdots & \sigma_{A_N,M} \end{pmatrix} \quad (3.10)$$

$$F = \begin{pmatrix} F_{A_1,1} & F_{A_2,1} & \cdots & F_{A_N,1} \\ F_{A_1,2} & F_{A_2,2} & \cdots & F_{A_N,2} \\ \vdots & \vdots & \ddots & \vdots \\ F_{A_1,M} & F_{A_2,M} & \cdots & F_{A_N,M} \end{pmatrix} \quad (3.11)$$

Substituindo-se as matrizes de média e desvio padrão do tempo de execução e de frequência de ocorrência de repetição, Equações 3.9, 3.10 e 3.11, respectivamente, na função de distribuição de probabilidade dada pela Equação 3.4, é possível construir uma matriz de probabilidade. Todas as operações entre matrizes são realizadas utilizando operadores de Hadamard (HORN, 1990) ou operações termo-a-termo.

Por convenção matemática, e para evitar um problema de indeterminação no cálculo de probabilidade associada, uma ação cuja repetição não ocorre nas amostras encontradas, deve ter o valor de média não-nulo e o valor de desvio padrão deve ser infinito. Por convenção, neste trabalho utiliza-se o valor de média 0.0. Esta escolha foi feita para que a função distribuição de probabilidade seja avaliada com valor 1.0 para ações inexistentes na amostra de treinamento, independente da ação ser executada pelo jogador numa *build order* utilizada no processo de classificação.

$$p(t) = \begin{pmatrix} F_{A_1,1} e^{-\frac{(t_{A_1,1} - \mu_{A_1,1})^2}{\sigma_{A_1,1}^2}} & F_{A_2,1} e^{-\frac{(t_{A_2,1} - \mu_{A_2,1})^2}{\sigma_{A_2,1}^2}} & \cdots & F_{A_N,1} e^{-\frac{(t_{A_N,1} - \mu_{A_N,1})^2}{\sigma_{A_N,1}^2}} \\ F_{A_1,2} e^{-\frac{(t_{A_1,2} - \mu_{A_1,2})^2}{\sigma_{A_1,2}^2}} & F_{A_2,2} e^{-\frac{(t_{A_2,2} - \mu_{A_2,2})^2}{\sigma_{A_2,2}^2}} & \cdots & F_{A_N,2} e^{-\frac{(t_{A_N,2} - \mu_{A_N,2})^2}{\sigma_{A_N,2}^2}} \\ \vdots & \vdots & \ddots & \vdots \\ F_{A_1,M} e^{-\frac{(t_{A_1,M} - \mu_{A_1,M})^2}{\sigma_{A_1,M}^2}} & F_{A_2,M} e^{-\frac{(t_{A_2,M} - \mu_{A_2,M})^2}{\sigma_{A_2,M}^2}} & \cdots & F_{A_N,M} e^{-\frac{(t_{A_N,M} - \mu_{A_N,M})^2}{\sigma_{A_N,M}^2}} \end{pmatrix} \quad (3.12)$$

onde  $t$  é uma matriz que indica o tempo de execução dos pares ação-repetição de uma *build order* que se deseja classificar.

### 3.4 Classificação

O processo de classificação é realizado de forma independente para cada *label* (*build order*) utilizado no processo de treinamento. Os valores de probabilidade para cada par de ação-tempo de um *replay* são multiplicados e o valor final determina a semelhança de uma *build order* treinada e a executada.

Considerando-se o exemplo anterior, é possível aplicar o instante de execução de cada ação do jogador no vetor de ações correspondente e obter a probabilidade de que uma ação qualquer  $A$ . Dessa forma, a probabilidade total de que um conjunto de ações seja parte de uma *build order* qualquer é dada pela Equação 3.13:

$$p_{BO} = \prod_{n,i} p(BO|A_{(n,i)}, t) \quad (3.13)$$

onde  $n$  indica o tipo da ação,  $i$  a sua repetição e  $p(BO|A_{(n,i)}, t)$  é a probabilidade de que a ação seja parte de uma *build order*  $BO$  dado que  $A_{(n,i)}$  foi executada no instante de tempo  $t$ , conforme a Equação 3.4.

#### 3.4.1 Forma Matricial

Dada a matriz de probabilidades de uma *build order* treinada, conforme a Equação 3.12, a probabilidade final de uma *build order* treinada é dada pela redução da matriz através da operação de multiplicação de seus termos, conforme Equação 3.14.

$$p_{BO} = \prod_{i,j} p(t)_{ij} \quad (3.14)$$

onde  $i$  e  $j$  são respectivamente as linhas e as colunas das matrizes de probabilidade e de tempo de execução da *build order* a ser classificada,  $t$  é a matriz de tempo de execução da *build order* para ser classificada,  $p$  é a matriz de probabilidades de uma *build order* qualquer  $BO$  e  $p_{BO}$  é a probabilidade reduzida da matriz  $p$  cujo valor é equivalente à operação da Equação 3.13.

### 3.5 Implementação

A implementação do método foi feita em Python 2.7 utilizando a biblioteca *s2protocol* (Blizzard Entertainment, Inc., 2013) para processamento dos arquivos de *replay*, disponibilizada em licença de código fonte aberto no GitHub.

Para realizar a extração dos eventos do uma função chamada *parse()* foi definida. Esta função é responsável por extrair todas a *build orders* de todos jogadores que par-

ticiparam na partida contida num arquivo de *replay*. A função pode ser dividida em 3 etapas.

Na primeira etapa, extrai-se do cabeçalho do arquivo informações que associam um identificador inteiro, denominado de *player ID* com o nome do jogador e a raça. O identificador inteiro é usado nas estruturas de dados dos eventos para associar o evento com um jogador. Com estes dados, é construído um dicionário contendo as seguintes informações.

```
players = {
    playerID: {
        'Name':      ..., // nome do jogador
        'BuildOrder': ..., // estrutura que lista as ações da build order
        'Race':      ...  // nome da raça do jogador
    }
}
```

Depois de criada a estrutura de dados do jogador, é possível realizar a extração de cada evento do *replay*. Para cada estrutura de dados de evento recebida pela biblioteca de processamento conforme o exemplo abaixo:

```
{ '_bits': 296,
  '_event': 'NNet.Replay.Tracker.SUnitBornEvent',
  '_eventid': 1,
  '_gameloop': 2467,
  'm_controlPlayerId': 2,
  'm_unitTagIndex': 261,
  'm_unitTagRecycle': 1,
  'm_unitTypeName': 'Adept',
  'm_upkeepPlayerId': 2,
  'm_x': 34,
  'm_y': 152 }
```

Antes de processar qualquer elemento do evento, converte-se o campo `__gameloop` que representa o número de loops de jogo transcorridos até o momento do evento. A conversão para tempo em segundos é feita utilizando a expressão da Equação 3.1. Se o tempo de execução é superior à 6 minutos, o evento é descartado.

Caso o tempo seja inferior ou igual à 6 minutos, checka-se o tipo de evento. Se o evento for do tipo **SUnitBornEvent**, **SUnitInitEvent** ou **SUpgradeEvent** extrai-se alguns atributos da sua estrutura de evento. Caso contrário, o evento é descartado.

Se o evento for do tipo **SUnitBornEvent** ou **SUnitInitEvent**, então extrai-se do campo `m_unitTypeName` o nome da unidade treinada e a ID do jogador que executou



a ação, contida em `m_upkeepPlayerId`. A ID é usada para obter a estrutura de build order apresentada anteriormente e uma ação com nome `m_unitTypeName` e tempo de execução correspondente à `_gameloop` é adicionada na estrutura de dados de build order do jogador.

Se o evento for do tipo **SUpgradeEvent**, então extrai-se do campo `m_upgradeTypeName` o nome do melhoramento executado e a ID do jogador que executou a ação, contida em `m_playerId`. A ID é usada para obter a estrutura de build order apresentada anteriormente e uma ação com nome `m_upgradeTypeName` e tempo de execução correspondente à `_gameloop` é adicionada na estrutura de dados de build order do jogador.

Nem todos valores de `m_unitTypeName` ou `m_upgradeTypeName` são válidos e é necessário checar em uma lista de valores permitidos antes que os valores sejam adicionados à estrutura de dados de build order do jogador. A lista de todas ações permitidas estão no Anexo A.

É importante observar que não há sobreposição entre os nomes de unidades treinadas em `m_unitTypeName` e os nomes dos melhoramentos feitos em `m_upgradeTypeName` e portanto os próprios nomes são utilizados como ações. Isto é, conforme o exemplo dado, a sua ação correspondente no processo de treinamento do método é chamada de Adept pois é o valor contido no campo `m_unitTypeName`.

Após a construção da estrutura de dados de uma *build order*, é possível executar o algoritmo de treinamento. Este algoritmo é responsável pelo cálculo de média e desvio padrão de cada ação coletada na etapa anterior. O cálculo é feito conforme descrito na seção 3.3.

O classificador foi implementado numa classe em Python chamada de *BuildOrder-Matcher*. Esta classe é responsável pela organização das estruturas de dados do treinamento e classificação das *build orders*.

O processo de treinamento foi implementado num método denominado *train()* na classe *BuildOrderMatcher* onde é dado um conjunto de estruturas de *build order*, conforme extraído pelo método descrito na função *parse()*, e um *label* que identifica o nome que representa o conjunto de *build orders* dado. A função de treino itera sobre cada uma das *build orders* dadas e acumula o tempo de execução de cada uma das ações em termos das suas repetições em uma lista. A repetição é definida como a sequência em que uma mesma ação é executada. Se uma ação é executada ao total 3 vezes numa *build order*, então a primeira vez que ela é executada é definida como a repetição 1, a segunda vez que ela é executada é definida como a repetição 2 e a terceira como a repetição 3. Os valores de tempo de cada par ação-repetição são adicionados em uma estrutura de dicionário com 3 níveis. O primeiro nível representa o *label* de treinamento, o segundo nível representa o nome a ação e o terceiro nível representa a repetição.

Após a adição de todas *build orders* e seus *labels* pelo método *train()* da classe *BuildOrderMatcher*, um segundo método *build\_distribution()* deve ser invocado para que sejam geradas as funções distribuição de probabilidade conforme a Equação 3.4. Esta função é responsável pelo cálculo dos parâmetros da função distribuição de probabilidade. Um conjunto distinta de atributos da função distribuição de probabilidade é calculado e associado a cada par ação-repetição para cada *label* dado no método *train()*.

A função *build\_distribution()* itera por repetição de cada ação de cada um dos labels dados em *train()* e verifica que o par ação-repetição ocorre em pelo menos 2 amostras, caso contrário não seria possível atribuir um valor de desvio padrão para o par. Se há ao menos 2 elementos, calcula-se o parâmetro  $A$  da Equação 3.4 pelo tamanho de elementos (valores de tempo em segundos em que cada amostra executou uma ação, se uma ação não foi executada em alguma amostra, a lista possui um item a menos). O parâmetro é calculado dividindo-se o número de elementos de tempo na lista do par ação-repetição pelo número total de *build orders* dada para treinamento. De forma que, se um par ação-repetição ocorre em todas as amostras, seu valor é de 1.0 ou se um par ação-repetição ocorre em apenas duas de dez amostras de treinamento, seu valor é de 0.2.

Por fim, calcula-se o valor de média e desvio padrão do tempo de execução de cada par e constrói-se um objeto que representa de forma abstraída a função distribuição de probabilidade utilizando os parâmetros  $A$ ,  $\mu$  e  $\sigma$ . Ao invocar o método *apply()* com um valor de tempo  $t$ , é retornado o valor correspondente da função distribuição de probabilidade.

```
class NormalDistributionFunction:
    def __init__(self, mean, sigma=1.0, amplitude=1.0):
        self.mean = mean
        self.sigma = sigma
        self.amplitude = amplitude

    def apply(self, t):
        if t is None:
            return 0.0
        return self.amplitude * math.exp(-(math.pow(t-self.mean, 2) /
            (2 * math.pow(self.sigma, 2))))
```

O objeto representando a função distribuição de probabilidade é adicionado à uma estrutura de dicionário com 3 níveis. O primeiro nível representa o *label* de treinamento, o segundo nível representa o nome a ação e o terceiro nível representa a repetição.

Após a chamada do método *build\_distribution()* a classe está pronta para classificação de outras *build orders* desconhecidas através do método *classify()* onde é dado a estrutura de dados de uma *build order* desconhecida e o algoritmo calcula os valores de probabilidade associada à cada *label*. O cálculo é feito conforme descrito na seção 3.4.

O método *classify()* itera nos *labels* de treinamento e itera sobre a sequência de ações executadas na *build order* dada. Um contador é mantido por *label* e por ação que indexa o índice de repetição que é incrementado a cada ocorrência de uma ação com mesmo nome. Com isto, dado o *label* da iteração, a ação da iteração e o índice da repetição da ação, pode-se utilizar a função distribuição de probabilidade e calcular o valor de probabilidade associado ao par ação-repetição. O valor individual de probabilidade de cada par é multiplicado pelo valor total de probabilidade do *label*. A função retorna um dicionário indexado pelo *label* e cujo valor é produto das probabilidades.

A seleção da *build order* correta é feita escolhendo o *label* que possui o maior valor de probabilidade agregada.

## 4 Resultados e Discussões

No processo de treinamento, é extraído a sequência de construção de cada jogador (a *build order*), calculadas estatísticas de primeira ordem (média e desvio padrão) e a frequência de cada par ação-repetição em relação ao conjunto de replays correspondentes a mesma *build order*.

No processo de classificação, as probabilidades individuais de cada ação executada são calculadas e multiplicadas para obter um número que indica a probabilidade de que uma dado conjunto de ações pertença à uma *build order* conhecida.

O método foi treinado utilizando 7 build order da raça Protoss.

**Adept Glaives** : 6 replays

**Adept Stargate** : 14 replays

**Adept Immortal** : 13 replays

**Stalker Immortal** : 3 replays

**Adept Prism DT** : 5 replays

**Stalker Disruptor** : 5 replays

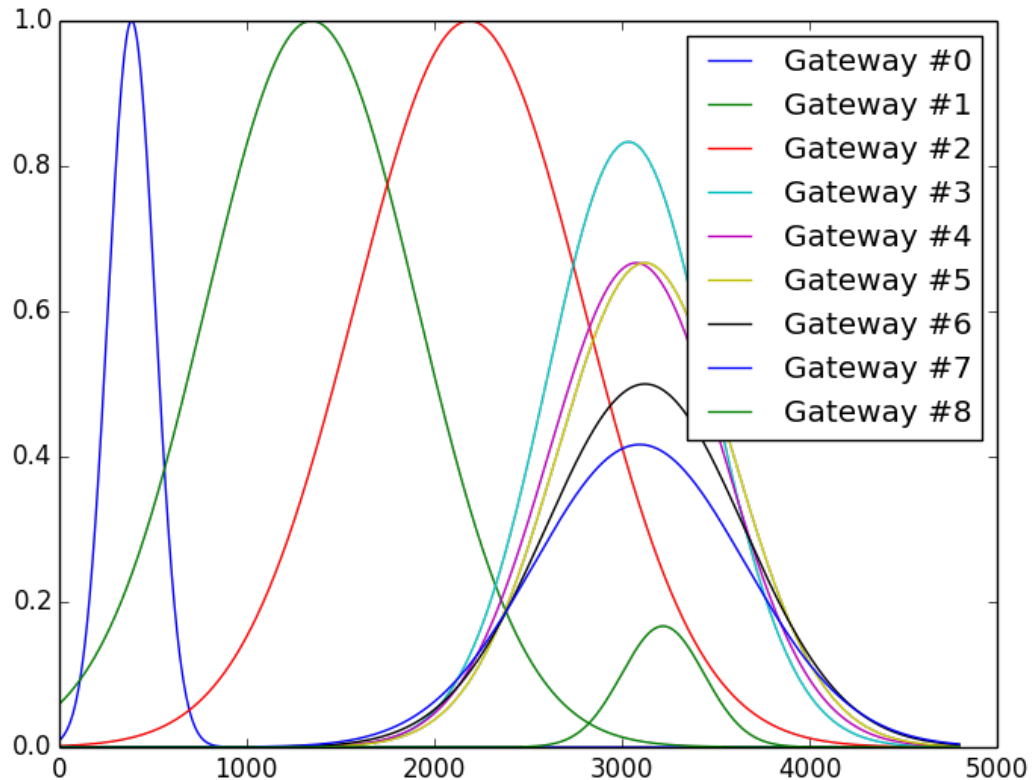
**Blink Stalker** : 8 replays

A Figura 4 apresenta um exemplo das funções distribuição de probabilidade para uma ação do tipo "Gateway" da *build order* "Adept Immortal".

Observa-se que a frequência de ocorrência das repetições no *dataset* de treinamento é decrescente. Isto é uma consequência do método escolhido para classificação. A primeira execução de uma ação, irá, independentemente do tempo em que for executada, considerada como a primeira repetição. A redução da frequência tem duas causas principais:

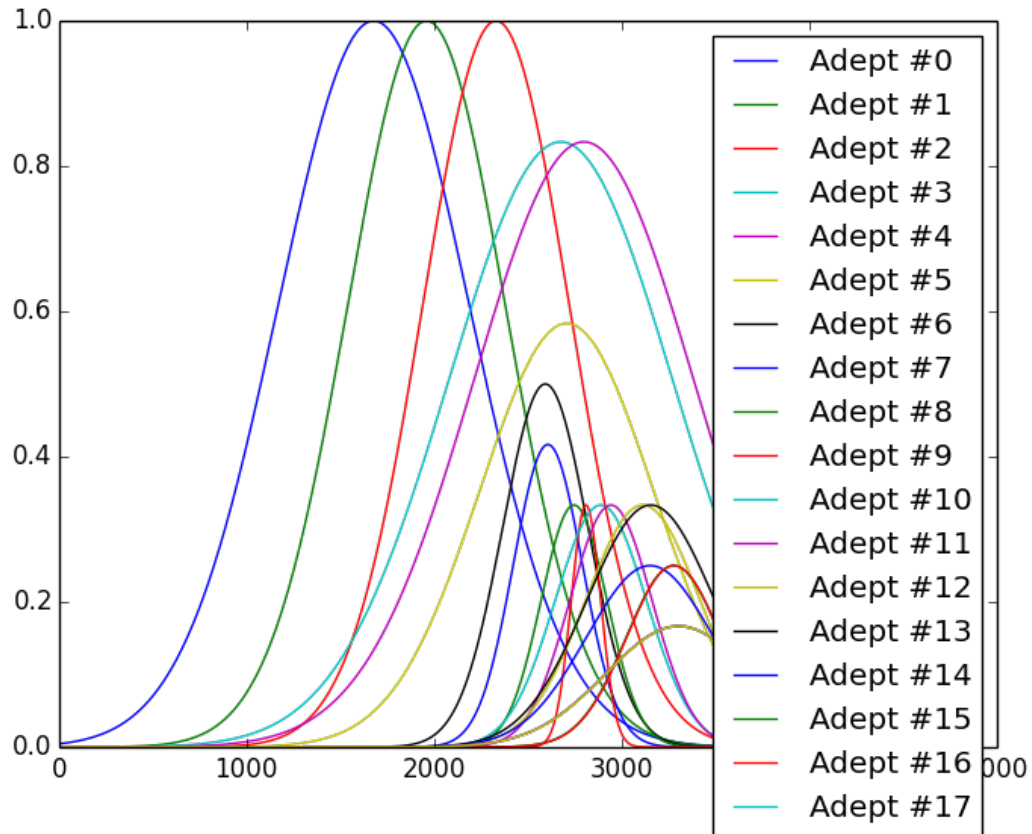
**Alteração da estratégia do jogador** : um jogador pode ter escolhido uma estratégia levemente diferente devido ao contexto do jogo ou em resposta à *build order* do oponente;

**Ações opcionais** : algumas ações podem ser opcionais e somente são executadas em alguns mapas.

Figura 4 – Forma gráfica de distribuição estatística de uma ação *Gateway* e suas repetições

Há uma tendência no desvio-padrão e variância do tempo de execução do par ação-repetição de crescerem ao decorrer da partida. Os principais motivos deste crescimento é a incapacidade de jogadores humanos executarem as ações de forma perfeita e sem qualquer variabilidade. Este fator é agravado com a característica da "árvore tecnológica" onde algumas ações possuem dependências tecnológicas em outras anteriores. Embora a tendência seja visível na maior parte das ações, ainda é possível que ela reduza em alguns casos específicos.

Um exemplo onde o desvio padrão reduz de forma significativa é visível na Figura 5, a primeira repetição possui um desvio padrão significativamente maior ao da segunda repetição. Este comportamento é esperado quando a primeira ação é considerada opcional e incorre num erro onde a primeira repetição deveria ter frequência menor que a segunda. Infelizmente, devido à forma de como o método de separação foi estabelecido, não é possível evitar este efeito.

Figura 5 – Forma gráfica de distribuição estatística de uma ação *Adept* e suas repetições

## 4.1 Autoclassificação

Após o treinamento, foi calculado o índice de "autoclassificação" que é a taxa de acerto do método ao classificar os mesmos *replays* utilizados no treinamento. Os resultados deste teste estão expostos na Tabela 2.

Tabela 2 – Resultado da autoclassificação

<i>Build order</i>	Acertos	Total	Taxa de acerto
<b>Adept Glaives</b>	6	6	1.0
<b>Adept Stargate</b>	7	14	0.5
<b>Adept Immortal</b>	11	13	0.85
<b>Stalker Immortal</b>	3	3	1.0
<b>Adept Prism DT</b>	5	5	1.0
<b>Stalker Disruptor</b>	2	5	0.4
<b>Blink Stalker</b>	6	8	0.75

Estes resultados apontam para um erro de classificação alto para as *build orders* "Adept Stargate", que engloba 3 *build orders* diferentes, sob uma sequência de ações iniciais

semelhantes. A alta taxa de erro é justificada pela frequência de ocorrência das últimas ações com 3 *branches* possíveis, implica na redução do valor da função distribuição de probabilidade do par ação-repetição. A taxa de erro de "Stalker Disruptor" é justificada pelo fato das duas builds "Stalker Disruptor" e "Stalker Immortal" serem muito semelhantes, a divergência entre as duas ocorre muito próximo do marco de 6 minutos onde as ações foram truncadas uma vez que todos os erros de classificação da *build order* "Stalker Disruptor" apontam para "Stalker Immortal".

## 4.2 Validação cruzada

A fim de realizar a validação do resultado, utilizou-se um *dataset* individual de *replays* que não foram utilizados no treinamento para validar os resultados do método. O número de *replays* foi limitado pela capacidade manual de classificação. Neste dataset, um número variável de *replays* foi utilizado.

Tabela 3 – Resultado da validação cruzada

<i>Build order</i>	Acertos	Total	Taxa de acerto
<b>Adept Glaives</b>	3	3	1.0
<b>Adept Stargate</b>	3	4	0.75
<b>Adept Immortal</b>	10	11	0.90
<b>Stalker Immortal</b>	6	7	0.85
<b>Adept Prism DT</b>	1	1	1.0
<b>Stalker Disruptor</b>	14	16	0.875
<b>Blink Stalker</b>	10	12	0.83

## 4.3 Teste de ruído

Para testar a robustez do método sob condições de informações incompletas, como seria necessário caso uma inteligência artificial (AI) estivesse testando as possibilidades de ações de um jogador humano.

Neste teste, foi removido de forma aleatória um percentual de ações de cada *build order* do *dataset* de validação. Para que o teste seja replicável, utilizou-se o nome do replay (conforme indicado no índice dos *replays* do *dataset* de treinamento) como *seed* para a geração da sequência de exclusão das ações. Isto garante que independente de onde e quem execute o algoritmo, o resultado será o mesmo.

Tabela 4 – Resultado da validação cruzada

<i>Build order</i>	2%	5%	10%	20%	30%	50%	70%	80%
<b>Stalker Immortal</b>	0.86	0.57	0.43	0.14	0.29	0.14	0.29	0.43
<b>Adept Glaives</b>	1.00	1.00	0.33	0.00	0.00	0.00	0.00	0.00
<b>Adept Stargate</b>	0.75	0.75	0.50	0.25	0.50	0.00	0.00	0.00
<b>Stalker Disruptor</b>	0.88	0.69	0.38	0.06	0.00	0.00	0.00	0.00
<b>Blink Stalker</b>	0.83	0.67	0.75	0.50	0.42	0.33	0.25	0.17
<b>Adept Immortal</b>	0.91	0.73	0.73	0.64	0.55	0.27	0.00	0.00
<b>Adept Prism DT</b>	1.00	1.00	0.00	1.00	1.00	0.00	0.00	0.00

No teste de ruído é observável que algumas das *build orders* onde o classificador obteve o melhor desempenho, como é o caso da "Adept Prism DT", o classificador também foi capaz de suportar maior nível de ruído. Isto se deve provavelmente ao fato desta *build order* em especial, ser bastante diferente das demais.



## 5 Conclusões

Conforme visto no teste de autoclassificação, o método possui dificuldade em classificar *build orders* muito genéricas onde há alta possibilidade de variação, conforme é o caso da *build order* "Adept Stargate". No teste de validação cruzada, o método foi capaz de conseguir uma taxa de aproximadamente 85% de acerto, chegando a 100% de acertos em alguns casos específicos, embora com poucas amostras.

*build orders* que possuem várias ações opcionais ou alternativas, como é o caso de "Adept Stargate" onde próximo dos 5 minutos de jogo pode ser escolhido entre 3 caminhos possíveis, apresentaram problemas de classificação devido a ponderação de probabilidade devido a frequência de ocorrência das ações. Como as ações alternativas ao final da *build order* não ocorrem em todas as amostras e, portanto, tem sua frequência abaixo de 1.0, o que implica na redução do valor de probabilidade ao final do método. Este efeito é benéfico, pois permite que sejam feitas classificações em vários níveis de detalhe. Por exemplo, seria possível a criação de um *label* para classificação utilizando uma forma mais genérica, como é o caso da "Adept Stargate" e outros *labels* para classificação individual para cada um de suas alternativas. Isto permita que seja possível de extrair algumas informações sobre uma *build order* desconhecida, mas que segue uma base comum à várias outras.

*build orders* semelhantes com pequena divergência ao final (próximo ao 6 minutos) apresentaram erros de classificação significantes, em especial "Stalker Disruptor" no teste de autoclassificação. Acredita-se que a causa desses erros é devido à grande variabilidade ao final da execução destas ações, inclusive alguns casos em que as ações passavam do limite de 6 minutos.

No teste de qualidade da classificação para simular condições de classificação em tempo real, foi adicionado de ruído (remoção de ações) para simular condizente com uma em que o jogo estivesse sendo jogado em tempo real, como seria caso um algoritmo de inteligência artificial estivesse precisando inferir informações sobre a estratégia de um jogador. Nestes testes, *build orders* mais distintas e características como "Adept Prism DT", onde a classificação mesmo com 30% conseguiu manter a taxa de acerto. Demais *build orders* perderam significativamente a taxa de acerto com o aumento de ruído.

## 6 Propostas de Trabalhos Futuros

- Utilização de um refinamento utilizando ações-chave

## Referências Bibliográficas

Blizzard Entertainment, Inc. *s2protocol*. 2013. Disponível em: <<https://github.com/Blizzard/s2protocol>>.

HORN, C. R. J. R. A. *Matrix Analysis*. 1st. ed. CUP, 1990. ISBN 0521305861, 0521386322. Disponível em: <<http://gen.lib.rus.ec/book/index.php?md5=03e9a9ebb67da1d5fa059774b3a12c4b>>.

SYNNAEVE, G.; BESSIERE, P. A bayesian model for opening prediction in rts games with application to starcraft. In: IEEE. *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. [S.l.], 2011. p. 281–288.

SYNNAEVE, G.; BESSIERE, P. A bayesian model for plan recognition in rts games applied to starcraft. *arXiv preprint arXiv:1111.3735*, 2011.

WEBER, B. G.; MATEAS, M. A data mining approach to strategy prediction. In: IEEE. *2009 IEEE Symposium on Computational Intelligence and Games*. [S.l.], 2009. p. 140–147.

# Glossário

**A | B | E | G | M | P | R | S | T | U**

## **A**

### **Adept**

unidade básica de exército da raça Protoss. 24, 34, *veja* Protoss

### **arvore tecnológica**

árvore de dependências que define pré-requisitos tecnológicos (melhoramentos ou estruturas) para aquisição de uma nova tecnologia.

## **B**

### **build order**

conjunto de ações ordenadas que formam a estratégia inicial de um jogador.

## **E**

### **estrutura**

uma construção (semelhante à um edifício) construído dentro do jogo com objetivo de treinar unidades, melhoramentos ou liberações da arvore tecnológica.

## **G**

### **Gateway**

estrutura de treinamento de unidades básicas terrestres da raça Protoss. *veja* Protoss

### **Glaives**

apelido para o melhoramento "Resonating Glaives". Aumenta a velocidade de ataque da unidade Adept aumentando significativamente seu poder bélico. *veja* Adept

### **gás**

recurso utilizado para unidades, estruturas ou melhoramentos mais avançados tecnologicamente.

## **M**

**melhoramento**

melhoramento de atributos de alguma unidade ou estrutura. Exemplo: aumento de velocidade de deslocamento, aumento de dano por ataque, aumento de velocidade de ataque. 16, 24, 34, 35, *veja* estrutura & unidade

**minério**

recurso mais básico e abundante do jogo. Requisito para todas unidades, estruturas ou melhoramentos.

**P****Protoss**

uma das 3 raças disponíveis no jogo.

**R****raça**

facção, espécie das unidades do jogador. Existem 3 raças disponíveis: Terran, Zerg e Protoss. Cada raça possui mecânicas, unidades, estruturas e melhoramentos completamente diferentes. 10, 11, 23, 27, 34, 35, *veja* Protoss

**recurso**

análogo à dinheiro ou matéria prima. Recursos básicos necessários para construir estruturas ou treinar unidades.

**repetição**

execução de uma ação múltiplas vezes.

**replay**

arquivo binário que contém todas ações realizadas durante um jogo, contém informações o suficiente para que seja possível reexecutar uma simulação do jogo caso seja necessário assistir o jogo. Também inclui algumas informações pré-processadas para uso de ferramentas externas..

**Robo**

apelido para "Robotics Bay", estrutura de treinamento de unidades terrestres avançadas da raça Protoss. *veja* Protoss

**S****Stargate**

estrutura de treinamento de unidades aéreas da raça Protoss. *veja* Protoss

**T****treinar**

ato de criar uma unidade durante o jogo. Requer que os pré-requisitos para cada unidade sejam fornecidos e que haja recursos o suficiente (minérios e gás).

**U****unidade**

soldado, personagem do jogo com objetivo militar.

## ANEXO A – Lista de nomes de ações utilizadas

- Adept
- AdeptPiercingAttack
- Archon
- Assimilator
- BlinkTech
- Carrier
- CarrierLaunchSpeedUpgrade
- Charge
- Colossus
- CyberneticsCore
- DarkShrine
- DarkTemplar
- Disruptor
- ExtendedThermalLance
- FleetBeacon
- Forge
- Gateway
- GraviticDrive
- HighTemplar
- Immortal
- Mothership
- MothershipCore
- Nexus
- Observer
- ObserverGraviticBooster
- Oracle
- Phoenix
- PhoenixRangeUpgrade
- PhotonCannon
- ProtossAirArmorsLevel1
- ProtossAirArmorsLevel2
- ProtossAirArmorsLevel3
- ProtossAirWeaponsLevel1
- ProtossAirWeaponsLevel2
- ProtossAirWeaponsLevel3
- ProtossGroundArmorsLevel1
- ProtossGroundArmorsLevel2
- ProtossGroundArmorsLevel3
- ProtossGroundWeaponsLevel1
- ProtossGroundWeaponsLevel2
- ProtossGroundWeaponsLevel3
- ProtossShieldsLevel1
- ProtossShieldsLevel2
- ProtossShieldsLevel3
- PsiStormTech

- 
- RoboticsBay
  - RoboticsFacility
  - Sentry
  - Stalker
  - Stargate
  - Tempest
  - TemplarArchive
  - TwilightCouncil
  - VoidRay
  - WarpGate
  - WarpGateResearch
  - WarpPrism
  - Zealot