

**Inteligência Artificial**

**Projeto Wumpus Agent**

Professor: Dr. Rodrigo Almeida Golçalves

Nome: André Garcia Dobermann

RA: 201610586

Curso: Engenharia da Computação

Sumário

[Introdução 3](#_Toc516188544)

[O problema 3](#_Toc516188545)

[Ferramentas 4](#_Toc516188546)

[Comunicação com o servidor 6](#_Toc516188547)

[Descrição da interface com o usuário 9](#_Toc516188548)

[Explicação dos símbolos 9](#_Toc516188549)

[Descrição da base de conhecimento 10](#_Toc516188550)

[Base de conhecimento 10](#_Toc516188551)

[Regras iniciais 10](#_Toc516188552)

[Regras implementadas 13](#_Toc516188553)

[Explicação da máquina de estados implementada pelo agente. 15](#_Toc516188554)

[Algoritmo de busca 15](#_Toc516188555)

[Resultados e observações 16](#_Toc516188556)

# Introdução

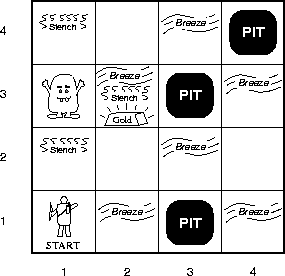
O projeto “Wumpus agent” foi proposto pelo professor Rodrigo Almeida Gonçalves na Facamp relacionada a matéria de inteligência artificial. O projeto consiste na criação de um agente inteligente que consiga resolver o problema do Wumpus, proposto inicialmente pelo livro Artificial Inteligence – A Modern Approach, Stuart Russell e Peter Norvig.

# O problema

O problema do Wumpus consiste basicamente em um jogo onde o jogador controla um explorador com o objetivo de achar ouro em um complexo de cavernas, que podem conter um abismo ou um monstro chamado Wumpus.

O complexo de cavernas é representado por uma grade de 10x10, em que cada célula representa uma caverna, cada caverna pode conter um abismo ou um Wumpus ou o ouro. O jogador pode andar somente em quadro direções, “NORTE”, “SUL”, “OESTE”, “LESTE”, ele também só consegue enxergar a caverna em que ele está, ou seja, o jogador não consegue olhar o que tem nas cavernas adjacentes, a não ser que ele se mova até elas. Porém o jogador pode deduzir o que existe nas cavernas adjacentes baseado no que ele percebe na caverna atual, pois as ameaças existentes como o Wumpus e o abismo possuem algumas características como: todas as cavernas adjacentes a uma caverna que existe um abismo, o jogador pode perceber uma brisa, assim como quando existe um Wumpus em uma caverna, todas as cavernas adjacentes o jogador pode sentir um cheiro ruim como pode ser observado na imagem 1.

Imagem 1: Exemplo de mapa 4x4



Fonte: voidexception.weebly.com/wumpus-world-agent.html

Com essas percepções o jogador pode deduzir o que existe nas cavernas adjacentes e assim ele pode tomar ações, como se mover para uma dessas cavernas, atirar uma flecha em uma dessas cavernas para matar o Wumpus ou arremessar uma pedra para saber se existe um abismo na caverna adjacente ou não.

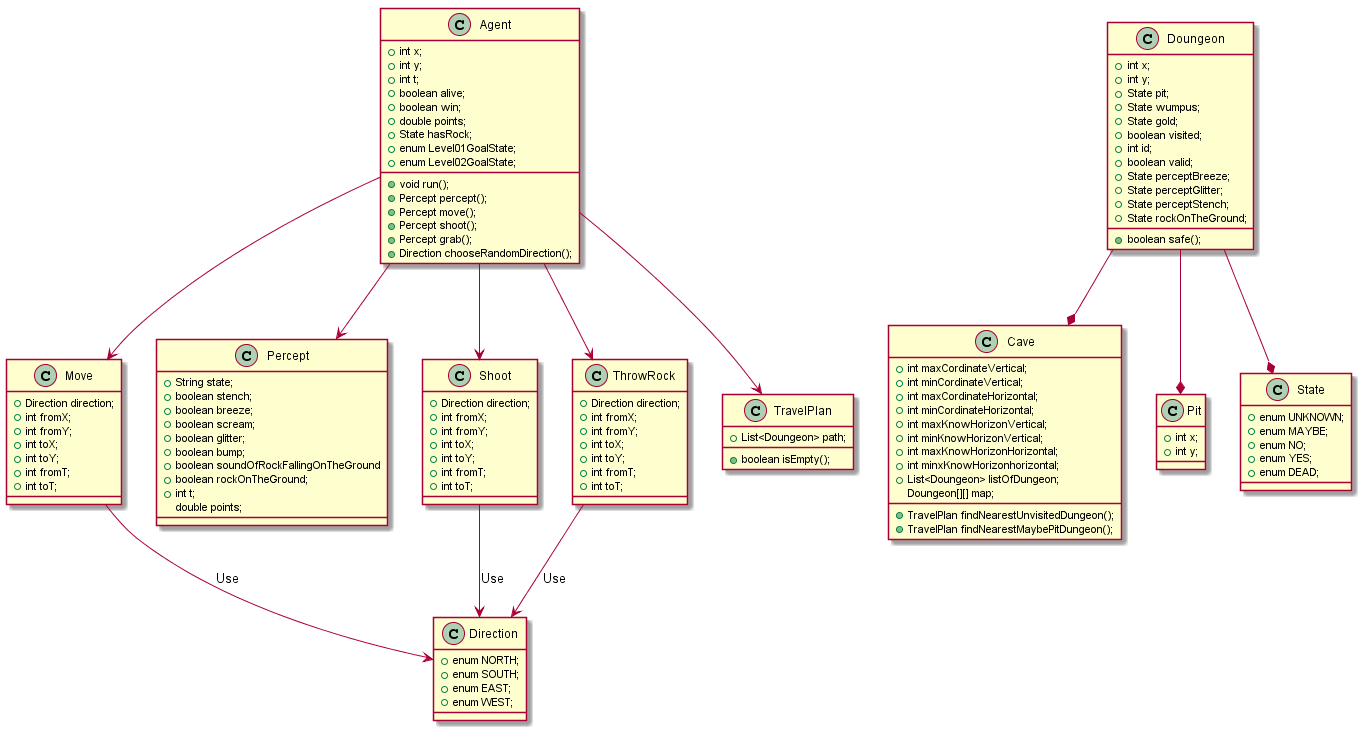
# Ferramentas

O jogo foi alocado em um servidor particular do professor Rodrigo, assim os alunos poderiam acessa-lo através do endereço “http://66.228.62.78:8080/server-0.5-SNAPSHOT/”. Para acessar inicialmente o servidor foi usado uma plataforma chamada Postman, que possui uma interface amigável e intuitiva para realizar operações com json, assim os alunos poderiam jogar o jogo e entender suas mecânicas e dificuldades, podendo tomar pensar em estratégias que ajudassem no desenvolvimento do agente.

Todos as classes foram escritas em java utilizando o NetBeans.

Os diagramas foram feitos usando a linguagem UML com uma extensão para o editor de texto atom.

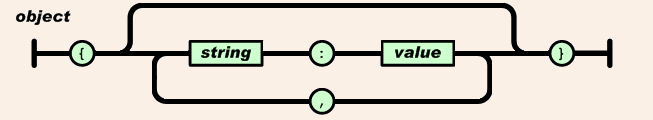
O controle de versionamento foi feito usando a plataforma github.



# Comunicação com o servidor

Para que o agente inteligente conseguisse conversar com o servidor foi utilizado o protocolo JSON, que é basicamente constituído por uma lista ordenada de valores, que pode ser considerado em outras linguagens como uma *array*, vetor e uma lista. O formato mais utilizado para envio de comandos foi o formato para *object* que possui o tipo do objeto enviado seguido pelo seu valor, como pode ser visto na imagem 2.

Imagem 2: Formato para *object* no protocolo JSON.



Fonte: json.org

Como já apresentado antes, o endereço necessário para solicitar informação do servidor é “http://66.228.62.78:8080/server-0.5-SNAPSHOT/”. Porém é necessário que o agente envie comandos junto com o endereço do servidor, que podem ser observados na tabela 1, onde cada comando necessita de um input.

O servidor precisa saber onde o usuário quer inserir o comando, portanto é necessário que após o endereço seja adicionado em qual área o comando deve ser executado. Para se criar um jogo a url deveria ser “http://66.228.62.78:8080/server-0.5-SNAPSHOT/creategame” seguido dos inputs que este comando necessita. Já para se realizar uma ação, como se mover, jogar uma pedra, agarrar o ouro ou agarrar a pedra é necessário que a url seja “http://66.228.62.78:8080/server-0.5-SNAPSHOT/action” seguido da ação que o agente deseja fazer. Vale ressaltar que para toda ação é preciso que o id da partida seja enviado junto para que o servidor saiba em qual partida ele deve executar o comando recebido. O diagrama 1 exemplifica um diagrama de sequência para uma conversa entre o agente inteligente e o servidor usando comandos como os explicados anteriormente.

Tabela 1: Comandos, inputs e outputs

|  |  |  |
| --- | --- | --- |
| Comandos | Inputs | Outputs |
| creategame | email:(Obrigatorio) lx:(Largura, default = 10) ly:(Altura,default = 10) pitdensity:(Porcentagem de pits) nWumpus:(Número de Wumpus, default = 3)  seed:(Número real, default = milissegundos do horário no servidor) | Id: identificação da partida |
| move | n | s | e | w | Percept após a ação |
| fire | n | s | e | w | Percept após a ação |
| Grab | grab gold | Percept após a ação |
| throwRock | n | s | e | w | Percept após a ação |
| grabRock | true (Porem aceita qualquer tipos de input) | Percept após a ação |

Fonte: Elaboração própria

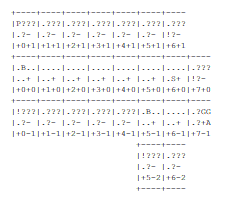


Fonte: Elaboração própria

Diagrama 1: Diagrama de sequência

# Descrição da interface com o usuário

Para que o usuário entenda o que está acontecendo foi criado uma notação para se representar as *dungeons*,que pode ser observada na imagem 3.



Onde cada retângulo representa uma *dungeon* e em cada uma delas existem 3 linhas com 4 símbolos cada, cada símbolo representa algo que existe relacionado a *dungeon* que o retângulo representa, cada símbolo possui um significado e uma posição dentro do retângulo.

## Explicação dos símbolos

P 🡪 A letra P é atribuída a primeira posição da primeira linha quando a *dungeon* que o retângulo representa é uma *dungeon* que possui um *pit*.

B 🡪 A letra B é atribuída a segunda posição da primeira linha quando o agente percebeu *breeze* nesta *dungeon*.

G 🡪 A letra G é atribuída na terceira e quarta posição da primeira linha, pois a terceira posição representa a *dungeon* que possui *glitter*  e a quarta posição representa o *gold*.

W 🡪 A letra W é atribuída na primeira posição da segunda linha e representa a *dungeon* que possui o Wumpus.

S 🡪 A letra S é atribuída na segunda posição da segunda linha, representa *dungeons* que possuem *stench.*

A 🡪 A letra A é atribuída na quarta posição e representa onde o agente está.

? 🡪 A interrogação pode aparecer em qualquer posição e representa incerteza em relação ao valor atribuído a posição em que ele se encontra.

A quarta posição da segunda linha pode possuir um + ou um -, onde o + sinaliza que o agente já visitou esta *dungeon* e o – sinaliza que ele não visitou,

A terceira linha possui dois números, podendo ser positivos ou negativos, eles representam as coordenadas da *dungeon* em um plano cartesiano, onde a origem é a *dungeon* onde o agente começou.

# Descrição da base de conhecimento

## 6.1- Base de conhecimento

A base de conhecimento é utilizada para armazenar as percepções recolhidas de sensores do meio ambiente, no caso do Wumpus a base de conhecimento armazena todas as percepções que o agente coleta, comparando depois com as regras existentes e como ele poderá agir diante do que o agente conhece e o como o ambiente se encontra.

## 6.2- Regras iniciais

As regras iniciais são as regras implementadas pelo professor Rodrigo.

A regra “*win\_the\_game*”, é acionada quando o agente está vivo e a caverna em que ele está possui *gold*, assim a regra faz com que o agente realize a ação *grab,* para pegar o *gold*, e modifica alguns atributos do agente, mudando o estado dele para “não vivo”, estado de que venceu e modifica os pontos somando a pontuação atual com a pontuação anterior acumulada.

A regra “*percept\_brezze*”, é acionada quando o agente não sabe se na caverna em que ele está existe brisa ou não, assim a regra busca saber se na resposta do servidor no t atual existe brisa ou não, caso exista ela insere na base de conhecimento que na posição atual existe brisa, caso contrário ela insere que na posição atual não existe brisa.

A regra “*percept\_stench*”, mostrada na imagem 4, é acionada do mesmo modo que a regra “*percept\_brezze*”, porem só é válida quando o agente não sabe se na posição atual existe *stench*, caso exista a regra insere na base de conhecimento que naquela posição existe brisa, caso contrário ela insere que não existe brisa na posição atual.

A regra “*percept\_glitter*”, mostrada na imagem 5, é acionada quando o agente não sabe se existe glitter na posição atual, caso a retorne que o *glitter* for igual a *true* a regra insere na base de conhecimento que existe *glitter* portanto existe *gold*, inserindo também na base de conhecimento que existe *gold* na posição atual, caso o servidor retorne “false” para glitter a regra insere na base de conhecimento que não existe *glitter* nem *gold* na posição atual.

A regra “*mark*\_*visited*\_*and*\_*register*\_*I*\_*survived*” é acionada quando a posição atual do agente está marcada como não visitada, então ela verifica se o estado do agente está ativo, e marca a posição x,y como uma posição sem pit nem Wumpus.

A regra “*realize\_that\_I\_moved\_without\_bumping*” é acionada quando o agente realizou um movimento e o servidor retorna *bump* igual a *false*, alegando que o agente não colidiu com uma parede após o movimento. Após ser acionada a regra insere a nova posição do agente na base de conhecimento e retira o movimento realizado.

A regra “*lost*” é acionada quando o *percept* no instante atual retornar que o estado é “LOST”, ou seja, que o agente morreu, seja por um pit ou seja por um Wumpus.

A regra ”*realize\_that\_I\_moved\_bumping*” é acionada quando o agente realizou um movimento e o *percept* retornou *bump* como *true*, ou seja, quando o agente se move e bate em uma parede. Após ser acionada a regra verifica qual direção que o agente se moveu e delimita as coordenadas máximas e mínimas baseada no movimento. Por exemplo, se o agente se moveu para cima e bateu na parede, ele está no valor máximo da coordenada y, assim ele sabe que não pode existir nenhuma caverna acima desse valor.

A regra “*removing\_invalid\_dungeons*” é acionada quando a coordenada de uma caverna não condizer com o tamanho do mapa, ou coordenadas máximas e mínimas, caso uma caverna possua valor do eixo x menos do que coordenada mínima, ela é dada como inválida, pois está fora do mapa.

A regra “*expand\_horizons\_north*” é acionada quando a caverna a norte do agente possui a coordenada y menor ou igual a coordenada máxima no eixo y, ou seja, se é possível ir para o norte, além disso a regra só é acionada se essa caverna ainda não foi explorada, o contrário acontece com a regra “*expand\_horizons\_south*”, porem nela é verificada se existe uma caverna ao sul da posição atual.

Assim como nas regras “*expand\_horizons\_north*” e “*expand\_horizons\_south*” as regras “*expand\_horizons\_east*” e “*expand\_horizons\_west*” verificam a existência de caverna em uma determinada direção, onde a regra “*expand\_horizons\_east*” verifica se existe uma caverna não explorada a leste da posição atual, e o contrário acontece com a regra “*expand\_horizons\_west*”, que verifica se existe uma caverna não explorada a oeste da posição atual. Assim essas regras adicionam as cavernas existentes na base de conhecimento.

A regra “*wumpus\_is\_dead*” é acionada quando a o *percept* diz que existe *stench* e se em alguma caverna adjacente o Wumpus está como estado *dead*, assim a regra modifica a caverna atual como sem *stench*.

A regra “*free\_of\_wumpus*” é acionada quando o agente não perceber *stench* e se as cavernas adjacentes estão com estado de “unknown” ou “maybe”, assim a regra pode mudar o estado de cada caverna adjacente para sem Wumpus. A mesma ideia é aplicada para a regra “*free\_of\_pit*” que modifica as cavernas adjacentes como sem pit, se na caverna atual não houver brisa e as adjacentes estiverem com estados de “unknow” ou “maybe”.

A regra “*maybe\_pit*” é acionada quando o agente perceber brisa e as cavernas adjacentes estiverem com estado de “unknown”, ou seja, se não existe nenhuma informação das cavernas adjacentes na base de conhecimento. O mesmo conceito é aplicado para a regra “*maybe\_wumpus*”, que só é acionada quando o agente perceber *stench* e não souber o que existe nas células adjacentes.

A regra “*explore\_neighborhood\_safe\_walk*” é acionada quando o agente estiver em “modo” de explorar a vizinhança e se ele não tiver realizado um movimento no instante anterior, além de as cavernas adjacentes não possuírem pit ou Wumpus e não ter sido visitada. Assim a regra cria um movimento para as cavernas adjacentes e as insere na base de conhecimento.

A regra “*change\_state\_to\_plan\_travel*” é acionada quando o estado atual do agente é “*explore\_neighbourhood*”, porém não existem cavernas adjacentes que ainda não tenham sido exploradas, assim a regra faz com que o agente mude seu estado para o estado de “*plan\_travel*”.

A regra “*step\_on\_travel\_to\_safe*” é acionada quando o agente não realizou nenhuma ação no t atual e seu plano de viagem está vazio. Então a regra retira o plano de viagem atual da base de conhecimento e insere um novo plano, verificando se existe um Wumpus no novo plano de viagem, caso exista, a regra insere um movimento para o agente realizar e atirar nas coordenadas do Wumpus, caso não exista, o agente apenas se move para onde não tenha visitado ainda.

A regra “*change\_state\_back\_to\_explore*” é acionada quando o estado atual do agente é “*explore\_neighborhood*”, não o agente não se moveu no instante anterior e o plano de viagem está vazio, assim a regra faz com que o plano de viagem atual seja retirado da base de conhecimento e o agente comece a explorar a vizinhança.

A regra “*plan\_travel*” é acionada quando o estado do agente for “*plan\_travel*”, ou seja, quando o agente estiver seguindo um plano de viagem. Assim que for acionada a regra irá verificar se existe alguma caverna que ainda não tenha sido visitada, se tiver ele fará o agente se mover até elas. Quando não houver mais cavernas visíveis que o agente ainda não visitou, o agente irá se mover para os lugares que talvez possuam Wumpus. Logo que não houver mais cavernas que talvez possuam Wumpus o agente irá verificar se existe algum Wumpus no mapa, caso exista ele irá mata-lo, do contrário ele irá até a caverna que talvez possua pit. Se já visitou todas as cavernas que podem possuir pit, ele faz um movimento aleatório.

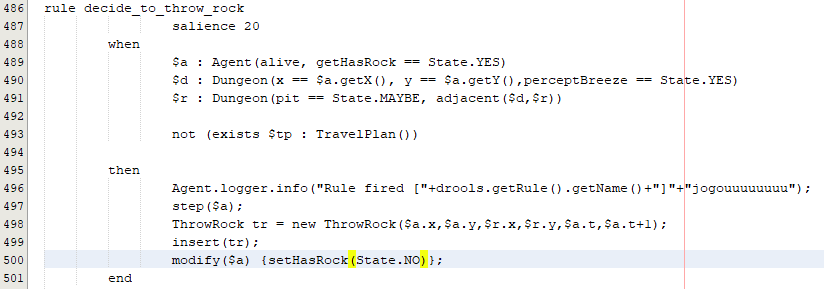
A regra “*realize\_shoot\_result*” é ativada quando o agente realizou um tiro em uma direção. Assim a regra retira o tiro da base de conhecimento, em seguida é feito um *percept* para saber se houve o grito do Wumpus, se houve o Wumpus é dado como morto na caverna em que o agente atirou, se não é possível ouvir o grito, nada acontece.

A regra “*shoot*” é ativada quando o agente atira em uma direção. Log em seguida a regra realiza um *percept* para saber o resultado do tiro que será avaliado pela regra “*realize\_shoot\_result*”.

A regra “*move\_agent*” é acionada assim que o agente realiza um movimento para uma caverna, para que então ela possa realizar um *percept* do resultado.

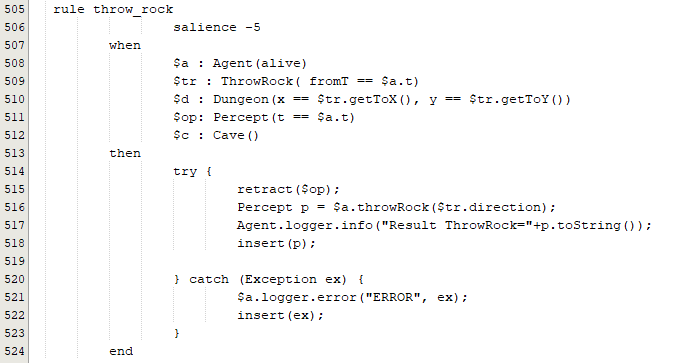
## 6.3- Regras implementadas

A primeira regra implementada foi a regra “*decide\_to\_throw\_rock*”,que pode ser vista na imagem 4 que é acionada quando o agente está em estado de explorar a vizinhança e possui uma pedra com sigo, esta regra faz com que o agente realize a ação de jogar uma pedra para saber se em uma *dungeon* adjacente possui ou não um *pit*.

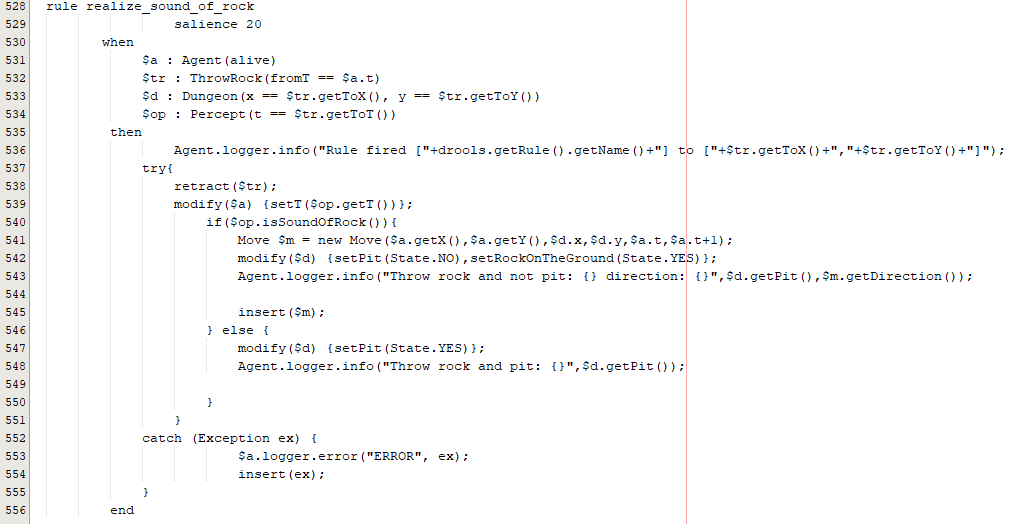
Imagem 4: Regra “*decide\_to\_throw\_rock*”

A regra “*throw\_rock*”, que pode ser vista na imagem 5 apenas percebe que o agente arremessou uma pedra e insere esta informação na base de conhecimento.

Imagem 5: Regra “*throw\_rock*”

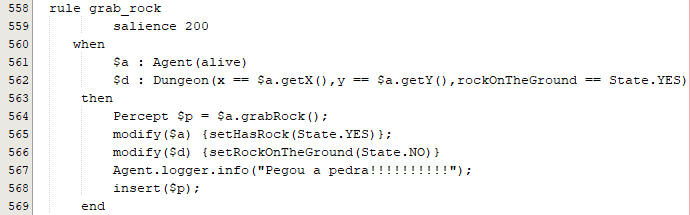


A regra “*realize\_sound\_of\_rock*”, que pode ser vista na imagem 6, é ativada quando no instante anterior o agente arremessou uma pedra, assim a regra analisa se o agente ouviu o som da pedra caindo no chão, significando que na *dungeon* em que ele jogou a pedra não possui um *pit* e então andar nesta direção, ou se ele não conseguiu ouvir o som da pedra caindo no chão, significando que existe um *pit* na posição em que o agente arremessou a pedra, podendo assim atualizar a base de conhecimento e modificando o estado daquela *doungeon* que foi mirada pelo agente.

Imagem 6: Regra ”*realize\_sound\_of\_rock*”

A regra “*grab\_rock”*, que pode ser vista na imagem 7, é acionada quando o agente consegue perceber que existe uma pedra na *dungeon* que ele está e se no último instante ele realizou a ação de jogar a pedra.

Imagem 7: Regra “*grab\_rock”*



## Explicação da máquina de estados implementada pelo agente.

O agente inteligente implementa uma máquina de estados chamada “Level02GoalState” que possui cinco estados, EXPLORE\_NEIGHBORHOOD, PLAN\_TRAVEL, TRAVEL\_TO\_SAFE, TRAVEL\_TO\_BEST\_WUMPUS. Cada estado determina qual a “prioridade” atual do agente, ou seja, qual o objetivo atual dele, seja matar um Wumpus, andar aleatoriamente ou simplesmente ir para algum lugar seguro.

O estado EXPLORE\_NEIGHBORHOOD faz com que o agente explore todas as *dungeons* que existem ao redor dele, até que alguma regra faça ele mudar de estado.

O estado PLAN\_TRAVEL é o estado onde o agente possui uma *dungeon* específica a qual ele quer visitar.

O estado TRAVEL\_TO\_SAFE faz com que o agente ande até a *dungeon* não visitada mais próxima

O estado TRAVEL\_TO\_BEST\_WUMPUS faz com que o agente viaje até próximo de alguma *dungeon* que talvez possua um Wumpus ou um *pit.*

## Algoritmo de busca

O programa possui um algoritmo de busca em largura (*braedth-first search*) implementado na classe cave, a escola para este algoritmo é pelo fato da sua fácil implementação e pela certeza de haver uma resposta, mesmo que não seja a mais otimizada.

Este algoritmo é usado pelo agente para descobrir um caminho para uma determinada *dungeon*, Wumpus mais próximo ou o pit mais próximo.

# Resultados e observações

Após terminar todas as alterações nas regras e classes, foi feito uma comparação do código sem alteração nenhuma (imagem 8) com o código em que houve alteração (imagem 9), nesta comparação. Ambos os códigos foram submetidos a 100 jogos idênticos, para que assim os dois pudessem ser analisados nas mesmas condições.

Imagem 8: Código sem alteração Imagem 9: Código com alteração

É possível observar uma melhora ao notar o gráfico 1 e calcular uma melhora de 1,17% em relação ao código sem nenhuma mudança, porém é um resultado bom levando em consideração a quantidade de regras que foram adicionadas.

## 9.1- Possíveis futuras implementações

Mesmo melhorando o desempenho após a adição de novas regras e de mudanças em algumas já existentes, há a possibilidade de se melhorar mais.

Uma das implementações não realizadas é a adição de uma variável na classe *Doungeon* chamada ”*pit*\_*probability*” e “*wumpus*\_*probability*”, em que seria calculada uma probabilidade de qual *doungeon* com o estado “maybe” tem mais chance de ser verdadeiro, ou seja, o agente poderia saber qual a *doungeon* que possui a maior probabilidade de se ter um *pit* ou um Wumpus e assim poder tomar decisões de jogar a pedra nas *doungeon* com menos probabilidade de *pit*, diminuindo as chances de jogar a pedra em um *pit* e perde-la.

A meta de melhoria é subir da média de 69% de vitórias para 80% de vitórias até a data de 30/6/2018.

# Considerações finais.

Tanto o código e este relatório se encontram no endereço https://github.com/andredob/Wumpus-Agent