# Desenvolvimento de um agente autónomo para o jogo Dig Dug

Licenciatura em Engenharia Informática

Inteligência Artificial

Docentes:

Diogo Gomes

Luís Seabra Lopes

Alunos:

Bárbara Nóbrega Galiza – 105937

Pedro Daniel Fidalgo de Pinho - 109986

universidade de aveiro

# Algoritmo

Utilizamos da pesquisa em árvore para descobrir o caminho mais eficiente ao inimigo mais próximo e de funções com testes (if – else) para avaliar as condições do ambiente.

Na pesquisa, implementamos uma versão semelhante à das aulas, com algumas modificações. A estratégia usada foi A*, por ser completa e garantir sempre uma solução óptima.

A cada "step" obtemos o inimigo mais próximo do digdug (distância euclidiana) e fazemos a pesquisa para achar o melhor caminho até ele. De seguida, obtemos a primeira posição do caminho calculado e testamos se o digdug pode se locomover para aquela posição.

**Descrição do algoritmo:**

1. Definir inimigo alvo (mais próximo) e guardar os inimigos vizinhos a ele (distâncias <=5)
2. Verificar se pode disparar
    1. Verificar se inimigo está no alcance da corda
    2. Verificar se inimigos vizinhos matarão o didug enquanto ele dispara
3. Calcular caminho até ao inimigo alvo
4. Decidir movimento
    1. Verificar se movimento matará o digdug e retornar outras opções de movimentos (keys) possíveis
    2. Escolher uma key aleatória dentre as retornadas (todas foram validadas para cada inimigo que apresenta risco)

# Métricas (I)

- Distância de "zona de perigo": Como mencionado, definimos a zona de perigo como 5 de distância (euclidiana) para obter os inimigos vizinhos ao inimigo alvo. Também utilizamos este valor para definir os inimigos perigosos próximos ao digdug, na verficação de movimento. Escolhemos este valor devido à possibilidade de morte por fogo do fygar, como exemplificado na matriz a seguir. Ao se mover pra posição "x", haveria o risco do Fygar se mover para a posição "y" e disparar um fogo de 3 casas. Por isso, incluímos a distância de 5 (posição "x" - posição "Fygar")

| | | | | | | |
|---|---|---|---|---|---|---|
| | DigDug | x | | | y | Fygar |
| | | | | | | |

- Heurística: A heurística implementada foi a distância euclidiana entre a posição do digdug e a posição do inimigo alvo. Optamos por fazê-la dessa forma porque a distância euclidiana é admissível e estima sempre uma distância muito próxima à distância real.

# Métricas (II)

- Função de custo: A nossa função de custo, assim como a heurística, é bastante simples.
  - Utilizamos custos bastante elevados para os nós de forma a evitar rochas, e caminhos que ultrapassassem as dimensões do mapa.
  - Para forçar o digdug a não cavar tantos túneis, utilizamos os valores [5,20], que foram escolhidos depois de testes com [0,5],[5,10] e [10,15], que apresentaram médias menores (de 3k a 5k pontos).
  - Porém, essa última verificação só é feita para distâncias menores que 3 para o inimigo (onde realmente importa), sendo o custo 0 em todos os outros casos, o que torna o algoritmo uma espécie de greedy (é uma combinação de greedy com A* devido as verificações de "bad path").
  - Também é necessário destacar que o custo é 0 para steps maiores ou iguais a 1000, a fim de corrigir um bug em que o digdug fica "preso" (anda para trás e para a frente) numa situação de um túnel horizontal paralelo ao túnel onde está o inimigo (o valor foi escolhido por ser atingido apenas em situações em que o didgdug estava "preso").

```python
def cost(self, action):
    x, y = action
    bad_path = 1000000

    # Preventing going outside of the map
    if x > self.size_x or y > self.size_y:
        return bad_path

    # Preventing choosing a path with a rock
    for rock in self.rocks:
        if rock.get("pos") == [x,y]:
            return bad_path

    if self.distance < 3 and self.step < 1000:
        if self.map[x][y] == 0:
            return 5
        else:
            return 20
    else:
        return 0  # greedy search
```

# Melhorias após primeira entrega

- Eliminação do bug "stuck":  Na entrega anterior, o nosso agente tinha um bug em que o digdug ficava "preso" em dar voltas próximo a um Fygar com inteligência "normal", pois este ia sempre para frente e pra trás e o digdug não ultrapassava a zona de perigo definida.
- Para resolver isso utilizamos uma verificação de número de steps: para fygars, se o jogo ultrapassar os 1000 steps, ou 700 steps mas o inimigo for o último do nível, o digdug abandona as verificações para as keys "a" e "d" (horizontal). Esses valores foram baseados na observação de quando era seguro definir uma situação de "stuck".
- Redefinição da heurística: Notamos que nossa heurística não era admissível, e por isso trocamos para a heurística descrita anteriormente.

- Comparação das médias:
- 1a entrega: Entre 30.7k (oficial) e 35k
- 2a entrega: ~40k

# Conclusão

Com esse projeto adquirimos conhecimentos sobre agentes inteligentes e algoritmos de pesquisa, e fomos capazes de implementar um agente inteligente capaz de jogar o jogo digdug com a aplicação destes conhecimentos.

Optamos por criar um agente mais reativo que deliberativo, visto que focamos mais na criação de verificações do que em criar custos e heurísticas elaboradas. Pensamos que uma abordagem mais equilibrada poderia trazer melhores resultados. Em uma futura implementação, tentaríamos utilizar melhor a árvore de pesquisa, e eliminaríamos a aleatoriedade na escolha de “key” alternativa.