Development of an autonomous agent for the game DigDug

- Guilherme Santos (107961)
- João Gaspar (107708)

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

Função agent_loop

- O ficheiro *student.py* é o componente que está conectado ao servidor do jogo e que traduz as soluções encontradas em movimentos do DigDug através das teclas;
- Quando um nível é apresentado ao *student*, é inicializada uma variável *flag* a 0 e o mesmo inicia um loop infinito onde continuamente recebe o estado do jogo do servidor. Se o nível do estado do jogo recebido é diferente do nível do estado do mapa atual, o *student* solicita um novo estado do mapa do servidor. É verificado então se o DigDug está presente no estado do jogo. Se estiver, o *student* atualiza a posição atual do agente e o estado do jogo. Em seguida, decide o próximo movimento com base na posição do DigDug, dos inimigos e das rochas no estado do jogo.
- Caso a posição atual do DigDug seja igual à posição anterior do DigDug a variável *flag* é incrementada e caso a mesma seja maior ou igual a 60 é selecionado um movimento aleatório para o DigDug executar.
- O movimento é então enviado de volta ao servidor. Este processo continua até que a conexão com o servidor seja fechada.

```python
async def agent_loop(server_address="localhost:8000", agent_name="student"):
    async with websockets.connect(f"ws://{server_address}/player") as websocket:
        await websocket.send(json.dumps({"cmd": "join", "name": agent_name}))

        agent = DigDugAgent()
        flag = 0
        old_pos = [0, 0]
        map_state = None

        while True:
            try:
                if(map_state is None):
                    map_state = json.loads(await websocket.recv())

                game_state = json.loads(await websocket.recv())

                if game_state["level"] != map_state["level"]:
                    map_state = json.loads(await websocket.recv())

                if("digdug" in game_state):
                    digdug = DigDug(game_state["digdug"])
                else:
                    continue

                agent.current_position = digdug
                agent.game_state = game_state

                action = agent.decide_movement(digdug, game_state["enemies"], game_state["rocks"])

                if(digdug.pos == old_pos):
                    flag += 1
                if flag < 60:
                    if action == 0:
                        key = "w"
                        flag = 0
                    elif action == 1:
                        key = "d"
                        flag = 0
                    elif action == 2:
                        key = "s"
                        flag = 0
                    elif action == 3:
                        key = "a"
                        flag = 0
                    elif action == 4:
                        key = "A"
                else:
                    key = random.choice(["w", "d", "s", "a"])
                    flag = 0

                old_pos = digdug.pos
                await websocket.send(json.dumps({"cmd": "key", "key": key}))
            except websockets.exceptions.ConnectionClosedOK:
                print("Server has cleanly disconnected us")
                return
```

Classes DigDug e DigDugAgent

- A classe DigDug é uma representação do personagem DigDug no jogo. Ela possui apenas um atributo, *pos*, que armazena a posição atual do DigDug no mapa. A posição é representada como uma lista de dois elementos, a coordenada x e a coordenada y, respetivamente;
- A classe DigDugAgent é onda a lógica do agente de inteligência artificial é implementada. Ela possui dois métodos estáticos, *find_closest_enemy* e *decide_movement*;
- O método *find_closest_enemy* recebe a instância do DigDug e a lista de inimigos como argumentos e retorna o inimigo mais próximo ao DigDug. Se a lista *enemies* não estiver vazia, ele usa a função *min* para encontrar o inimigo mais próximo.

```python
class DigDug:
    def __init__(self, pos):
        self.pos = pos

class DigDugAgent:
    @staticmethod
    def find_closest_enemy(digdug, enemies):
        if not enemies:
            return None

        closest_enemy = min(enemies, key=lambda enemy: math.dist(digdug.pos, enemy["pos"]))

        return closest_enemy
```

Classe DigDugAgent

- No método *decide_movement*, se existir um inimigo mais próximo, *closest_enemy*, a posição desse inimigo é armazenada na variável *enemy_pos* e as posições de todas as rochas são armazenadas na lista *rocks_pos*;
- Em seguida, o código percorre a lista de inimigos e, se encontrar um inimigo chamado Fygar, obtém as suas posições, x e y. Dependendo da sua direção, o código adiciona as posições onde o fogo pode atingir à lista *fires_pos*. Se a direção for 3 (esquerda), ele adiciona as posições à esquerda do Fygar. Se a direção for 1 (direita), ele adiciona as posições à direita do Fygar. Para cada direção, ele adiciona as posições até 6 espaços de distância na mesma linha y, bem como na linha y+1. Isto é feito para simular o fogo que o Fygar dispara;
- Em seguimento, o código ajusta a posição do inimigo mais próxima em relação à posição do DigDug. Se a direção do inimigo for 0 (cima) ou 2 (baixo), e a posição do DigDug for menor que a posição y do inimigo, a posição do inimigo é diminuída em 1. Se a posição y do DigDug for maior, ele aumenta a posição y do inimigo em 1. O mesmo acontece para a esquerda e direita;
- Em resumo, esta parte do código está a calcular as posições de perigo no jogo (fogo do Fygar e posições ao redor do Pooka quando está a atravessar paredes) e a ajustar a posição do inimigo mais próximo em relação à posição do DigDug.

```
if closest_enemy:
    enemy_pos = closest_enemy["pos"]
    rocks_pos = [x["pos"] for x in rocks]
    #print("ROCKS: ", rocks_pos)
    #ROCKS:  [[26, 5]]

    fires_pos = []
    for ene in enemies:
        #print("ENEMIES: ", enemies)
        #ENEMIES:  [{'name': 'Fygar', 'id': 'd4bc864e-483c-
        if(ene["name"] == "Fygar"):
            ene_x = ene["pos"][0]
            ene_y = ene["pos"][1]
            if(ene["dir"] == 3):
                fires_pos += [[ene_x, ene_y]]
                fires_pos += [[ene_x - 1, ene_y]]
                fires_pos += [[ene_x - 2, ene_y]]
                fires_pos += [[ene_x - 3, ene_y]]
                fires_pos += [[ene_x - 4, ene_y]]
                fires_pos += [[ene_x - 5, ene_y]]
                fires_pos += [[ene_x, ene_y+1]]
                fires_pos += [[ene_x - 1, ene_y+1]]
                fires_pos += [[ene_x - 2, ene_y+1]]
                fires_pos += [[ene_x - 3, ene_y+1]]
                fires_pos += [[ene_x - 4, ene_y+1]]
                fires_pos += [[ene_x - 5, ene_y+1]]
            elif(ene["dir"] == 1):
                fires_pos += [[ene_x, ene_y]]
                fires_pos += [[ene_x + 1, ene_y]]
                fires_pos += [[ene_x + 2, ene_y]]
                fires_pos += [[ene_x + 3, ene_y]]
                fires_pos += [[ene_x + 4, ene_y]]
                fires_pos += [[ene_x + 5, ene_y]]
                fires_pos += [[ene_x, ene_y+1]]
                fires_pos += [[ene_x + 1, ene_y+1]]
                fires_pos += [[ene_x + 2, ene_y+1]]
                fires_pos += [[ene_x + 3, ene_y+1]]
                fires_pos += [[ene_x + 4, ene_y+1]]
                fires_pos += [[ene_x + 5, ene_y+1]]
    if(closest_enemy["dir"] == 0 or closest_enemy["dir"] == 2):
        if(digdug.pos[1] < enemy_pos[1]):
            enemy_pos[1] -= 1
        elif(digdug.pos[1] > enemy_pos[1]):
            enemy_pos[1] += 1
    else:
        if(digdug.pos[0] < enemy_pos[0]):
            enemy_pos[0] -= 1
        elif(digdug.pos[0] > enemy_pos[0]):
            enemy_pos[0] += 1
```

Classe DigDugAgent

- É criado um dicionário de posições, onde cada chave representa as direções possíveis, e os valores representam as próximas posições respetivas;
- A prioridade do estilo de movimento (horizontal e vertical) é definido pela comparação das posições do DigDug e do inimigo mais próximo;
- Para cada par (direção:posição), é verificado primeiro se nenhumas das posições se encontra numa posição de perigo. Caso aconteça, as posições dessa direção e da direção oposta serão definidas com um valor alto ([9999, 9999]), para impedir um ciclo de movimento entre essas duas posições. Em seguida é verificado se as posições definidas se situam dentro do plano do jogo. Caso não se verifique, a posição dessa direção é definida também com um valor alto ([9999, 9999]);
- A partir do dicionário definido em cima, é criado um novo dicionário onde para cada direção, é calculada a distância a partir da posição dessa direção e da posição do inimigo; definimos de seguida uma distância de ataque para o DigDug, onde esta varia dependendo de onde o inimigo mais próximo se encontra e do número de inimigos ao redor do DigDug, assim pode garantir uma melhor segurança para ele;
- Por fim, se o inimigo estiver dentro da distância de ataque, o DigDug ataca. Se não houver nenhum inimigo dentro dessa distância o DigDug vai para a posição de menor custo. Caso existam 3 inimigos pertos do DigDug, ou o inimigo mais próximo seja um Pooka em modo *traverse* e que esteja perto do DigDug, este vai para a distância de maior custo.

```
new_pos = {}
if(closest_enemy["dir"] == 0 or closest_enemy["dir"] == 2):
    if digdug.pos[0] != enemy_pos[0]:
        new_pos[Direction.EAST] = [x+1, y]
        new_pos[Direction.WEST] = [x-1, y]
    else:
        new_pos[Direction.NORTH] = [x, y-1]
        new_pos[Direction.SOUTH] = [x, y+1]
else:
    if digdug.pos[1] != enemy_pos[1]:
        new_pos[Direction.NORTH] = [x, y-1]
        new_pos[Direction.SOUTH] = [x, y+1]
    else:
        new_pos[Direction.EAST] = [x+1, y]
        new_pos[Direction.WEST] = [x-1, y]

danger = 5
for pos in new_pos:
    if (new_pos[pos] in rocks_pos) or (new_pos[pos] in fires_pos) or (new_pos[pos] in pooka_trav_pos):
        danger = pos
        #print("DANGER: ",danger)
if danger != 5:
    #print("DANGER!")
    if (danger == 0 or danger == 2):
        new_pos[Direction.EAST] = [x+1, y]
        new_pos[Direction.WEST] = [x-1, y]
        new_pos[Direction.NORTH] = [9999, 9999]
        new_pos[Direction.SOUTH] = [9999, 9999]
    else:
        new_pos[Direction.NORTH] = [x, y-1]
        new_pos[Direction.SOUTH] = [x, y+1]
        new_pos[Direction.EAST] = [9999, 9999]
        new_pos[Direction.WEST] = [9999, 9999]

for pos in new_pos:
    if (new_pos[pos][0] < 0 or new_pos[pos][0] > 48) and (new_pos[pos] != [9999, 9999]):
        #print(new_pos[pos])
        new_pos[pos] = [9999, 9999]
        #print("LIMITE X")
    if (new_pos[pos][1] < 0 or new_pos[pos][1] > 24) and (new_pos[pos] != [9999, 9999]):
        new_pos[pos] = [9999, 9999]
        #print("LIMITE Y")
#print(new_pos)
distance = {}
for dir in new_pos:
    #print(new_pos)
    #{<Direction.NORTH: 0>: [1, 0], <Direction.SOUTH: 2>: [1, 2]}
    distance[dir] = math.dist(new_pos[dir], enemy_pos)

enemies_nearby = [ene for ene in enemies if math.dist(digdug.pos, ene["pos"]) <= 3]


if  x != enemy_pos[0] and y != enemy_pos[1] and len(enemies_nearby) < 3:
    attack_distance = 3
elif x != enemy_pos[0] and y != enemy_pos[1]:
    attack_distance = 2
else:
    attack_distance = 1

#print(enemies_nearby)
#print(closest_enemy)
#print(attack_distance)
if (math.dist(digdug.pos, enemy_pos) <= attack_distance) and len(enemies_nearby) < 3:
    return 4

if (len(enemies_nearby) >= 3) or ((closest_enemy in enemies_nearby) and (closest_enemy["name"] == "Pooka" and "traverse" in closest_enemy)):
    #print("REVERSED")
    return max(distance, key=distance.get)
else:
    return min(distance, key=distance.get)
```