UNIVERSIDADE FEDERAL DO RECÔNCAVO DA BAHIA - UFRB
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS - CETEC
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

TRABALHO DE INTELIGÊNCIA ARTIFICIAL - JOGO

CRUZ DAS ALMAS, AGO 2018

UNIVERSIDADE FEDERAL DO RECÔNCAVO DA BAHIA - UFRB
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS - CETEC
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

TRABALHO DE INTELIGÊNCIA ARTIFICIAL - JOGO

Relatório conferido ao componente curricular GCET159 – INTELIGÊNCIA ARTIFICIAL, ministrada pela professora Carolina Salcedo como avaliação parcial do referido componente. Elaborado por:

TAIRONE CONCEIÇÃO DIAS

CRUZ DAS ALMAS, AGO 2018

FUNDAMENTOS

A Inteligência Artificial (IA) é definida como uma série de técnicas e métodos que tem por objetivo programar em softwares a capacidade racional humana de resolver problemas. Em sua definição mais básica, um jogo eletrônico é considerado um software que promove a interação do jogador a fim de realizar escolhas e atingir objetivos em meio a regras pré-definidas. O uso de IA na industria de jogos começou com os games tradicionais, como xadrez, jogo da velha, entre outros. Atualmente é difícil encontrar um jogo que não utilize alguma técnica de IA em seu código. Algumas técnicas de implementação de IA, são: Máquinas de Estados Finito, Sistema Baseado em Regras, Lógica *Fuzzy*, Path-finding, Árvores de Decisão, Redes Neurais Artificiais.

Uma técnica muito utilizada em jogos é o sistemas baseados em regras (*Rule Based Systems* – RBSs) que podem ser utilizadas para representar comportamento de agentes. RBSs apresentam algumas vantagens como: (i) correspondem ao modo como as pessoas normalmente pensam sobre conhecimento, (ii) são bastante expressivos e permitem a modelagem de comportamentos complexos, (iii) modelam o conhecimento de uma maneira modular e (iv) são muito mais concisos que máquinas de estados finitos, além da facilidade na implementação.

Na técnica RBS, o conhecimento é definido através de um conjunto de parâmetros (variáveis) e um conjunto de regras que trabalham sobre esses parâmetros, de modo que durante a “tomada de decisão” essas regras são então processadas.

Seres humanos frequentemente analisam situações de maneiras imprecisas, isto é, fazem uso de termos como “pouca força”, “muito longe” ou “bastante apertado”. A lógica *fuzzy* consegue então representar problemas de uma maneira similar à maneira com que os seres humanos pensam sobre eles, pois conceitos como longe e pouco não são representados por intervalos discretos, mas por conjuntos *fuzzy* que permitem que um valor pertença a vários conjuntos com diversos graus de pertinência.

A lógica booleana tradicional só suporta valores de estado discretos resultando em mudanças de estado abruptas e para se conseguir respostas mais suaves em um sistema baseado em lógica booleana é necessária a adição de mais estados o que novamente pode levar a uma grande quantidade de regras, deixando o código extenso. Ao passo que lógica *fuzzy* evita esses problemas pois a resposta vai variar suavemente dado o grau de “verdade” das condições de entrada.

Em uma busca sem informação, ou cega, a informação sobre os estados são somente aquelas fornecidas na definição do problema. Então, somente geram sucessores e distinguem se um estado é objetivo ou não.

BATMAN vs CURINGA – O JOGO

Neste relatório apresentaremos um jogo com algumas técnicas de IA utilizadas na indústria de games. O objetivo do game é eliminar o(s) inimigo(s) antes que ele o elimine. Há um aumento da quantidade de inimigos em um determinado tempo e a medida que o jogador não consegue eliminar o(s) inimigo(s) a probabilidade do jogador perder o jogo é maior. Foi utilizado o Sistema Baseado em Regras (*Rule Based Systems* – RBSs) para o surgimento de inimigos bem como a sua quantidade, com um pouco de Lógica *Fuzzy* para a mudança de level e a busca cega para identificar um evento no jogo, como uma eliminação por exemplo. Como base para a implementação desse jogo, utilizou-se do módulo pygame.

Se o jogador levar um tempo maior para eliminar um inimigo, vai surgindo mais inimigos e a complexidade aumenta para tentar eliminá-los. O jogo proporciona duas formas de derrota para o jogador:

i. O inimigo pode lançar bombas em direção ao jogador, onde a quantidade é de forma aleatória, dada um proporção, bem como a direção da bomba.

ii. Os inimigos surgem no topo da tela e a cada movimento total no sentido horizontal ele vai descendo um nível no sentido vertical até chegar no mesmo nível do jogador, caso eles se choquem também acontece o *“game over”*.

Não há um *“You Win!”* neste modelo de jogo, o objetivo e conseguir a maior pontuação possível, levando o jogador a uma exaustão.

O jogo também não apresenta um histórico de pontuação realizada em jogadas anteriores, isso poderia ser realizado fazendo o uso de arquivo em Python. No entanto, apresenta level, onde a complexidade e a pontuação por cada inimigo eliminado aumentam.

Os personagens do jogo foram inspirados na clássica batalha dos quadrinhos entre Batman e Curinga, existem três agentes autômatos que representa o Curinga (inimigo) no jogo, cada um deles é diferenciado por uma imagem que se altera conforme a proximidade com o jogador (Batman). Eles se deslocam na horizontal e na vertical, onde este último, só acontece quando há um deslocamento total na horizontal, então ele desce um nível.

A classe base responsável por criar o(s) inimigo(s) e a atualização do deslocamento dele bem como todas as propriedades para renderizar a imagem é **class Curinga**, que contém as funções:

- __init__ : responsável por criar o(s) agente(s) na tela;
- update : atualiza a movimentação e a variação das imagens desse personagem.

Na Figura 1 é exibido o código referente a essa classe.

O personagem do Batman é controlado pelo agente humano (jogador), onde é possível apenas o deslocamento na horizontal, utilizando as setas do teclado (← e

→) e o lançamento de míssil, utilizando a tecla barra de espaço. A classe base responsável pela sua renderização e movimentação, podendo ser vista na Figura 2, é a **class Player** que contém as funções:

- __init__ : responsável por criar o agente na tela;
- move: redimensiona a imagem recebendo como entrada uma determinada posição;
- gunpos: retorna a posição no momento em que o jogador atirou.

Figura 1 – Classe Curinga

```
class Curinga(pygame.sprite.Sprite):
    vel = 13
    #taxa de mudanca das imagens entre os agentes automatos
    taxaMImg = 100
    images = []
    def __init__(self):
        pygame.sprite.Sprite.__init__(self, self.containers)
        self.image = self.images[0]
        self.rect = self.image.get_rect()
        self.facing = random.choice((-1,1)) * Curinga.vel
        self.frame = 0
        if self.facing < 0:
            self.rect.right = SCREENRECT.right

    def update(self):
        self.rect.move_ip(self.facing, 0)
        if not SCREENRECT.contains(self.rect):
            self.facing = -self.facing
            self.rect.top = self.rect.bottom + 1
            self.rect = self.rect.clamp(SCREENRECT)
        self.frame = self.frame + 1
        self.image = self.images[self.frame//self.taxaMImg%3]
```

Existem outras classes e funções necessárias para a dinâmica do jogo, entre elas:

- **Explosao, Tiro e Bomba.**
- **load_image, load_images, upLevelGame, levelScore e main.**

Nem todas as classes e funções estão explicadas neste relatório, no entanto, estão comentadas no código.

A função ***main***, é responsável pela dinâmica do jogo. Nesta classe criamos e configuramos todos os agentes, os eventos, o *display* e a taxa de atualização dos objetos na tela.

A ordem de aparecimento do Curinga bem como o *drop* de bombas é realizada de forma randômica multiplicada por um fator de probabilidade. Uma busca (busca cega) é realizada sobre os agentes, a cada atualização do frame, detectando uma

explosão, fazendo assim um incremento na pontuação do jogador, com base no RBSs e na lógica *Fuzzy*. O principal trecho de código que implementa esses conceitos de IA, podem ser vistas nas Figuras 3, 4 e 5.

Figura 2 – Classe Player

```
# Classe de um agente do jogo, neste caso, o jogador
class Player(pygame.sprite.Sprite):
    vel = 10
    bounce = 24
    gun_offset = -11
    images = []
    def __init__(self):
        pygame.sprite.Sprite.__init__(self, self.containers)
        self.image = self.images[0]
        self.rect = self.image.get_rect(midbottom=SCREENRECT.midbottom)
        self.reloading = 0
        self.origtop = self.rect.top
        self.facing = -1

    def move(self, direction):
        if direction: self.facing = direction
        self.rect.move_ip(direction*self.vel, 0)
        self.rect = self.rect.clamp(SCREENRECT)
        if direction < 0:
            self.image = self.images[0]
        elif direction > 0:
            self.image = self.images[1]
        self.rect.top = self.origtop - (self.rect.left//self.bounce%2)

    # retorna a posicao do agente no momento que atirou
    def gunpos(self):
        pos = self.facing*self.gun_offset + self.rect.centerx
        return pos, self.rect.top
```

Figura 3 – Busca de eventos no game

```
# Detectando explosao
for curinga in pygame.sprite.spritecollide(player, curingas, 1):
    Explosao(curinga)
    Explosao(player)
    levelScore()
    player.kill()

for curinga in pygame.sprite.groupcollide(tiros, curingas, 1, 1).keys():
    Explosao(curinga)
    levelScore()

for bomba in pygame.sprite.spritecollide(player, bombas, 1):
    Explosao(player)
    Explosao(bomba)
    player.kill()
```

Em um determinado nível de pontuação a dificuldade do jogo aumenta, onde as seguintes variáveis são alteradas: MAX_TIROS, CURINGA_ODDS e BOMBA_ODDS. Que são respectivamente: a quantidade máxima de tiros dados pelo jogador (Batman), probabilidade de aparecer um novo inimigo (Curinga) e a probabilidade de bombas caírem (Drop do Curinga).

Figura 4 – Função de aumento no nível do jogo

```
def upLevelGame():
    global MAX_TIROS, CURINGA_ODDS, BOMBA_ODDS, STR_LEVEL
    msg = "Level "+STR_LEVEL+": "+str(SCORE)+" pontos"
    pygame.display.set_caption(msg)
    if SCORE == 50:
        if MAX_TIROS > 0 and BOMBA_ODDS <= 100:
            MAX_TIROS = MAX_TIROS - 1
            CURINGA_ODDS = CURINGA_ODDS + 5
            BOMBA_ODDS = BOMBA_ODDS + 10
            STR_LEVEL = 'moderate'

    elif SCORE == 200:
        if MAX_TIROS > 0 and BOMBA_ODDS <= 100:
            MAX_TIROS = MAX_TIROS - 1
            CURINGA_ODDS = CURINGA_ODDS + 10
            BOMBA_ODDS = BOMBA_ODDS + 20
            STR_LEVEL = 'hard'

    elif SCORE == 400:
        if MAX_TIROS > 0 and BOMBA_ODDS <= 100:
            MAX_TIROS = 1
            CURINGA_ODDS = CURINGA_ODDS + 11
            BOMBA_ODDS = 100
            STR_LEVEL = 'insane'
```

Figura 5 – Função de aumento na pontuação do jogador

```
def levelScore():
    global SCORE
    if SCORE < 50 :
        SCORE = SCORE + 1
    elif SCORE < 200 :
        SCORE = SCORE + 2
    else :
        SCORE = SCORE + 5

    upLevelGame()
```

COMO JOGAR?

É necessário ter o Python instalado no computador e o módulo do pygame. Caso utilize o Linux ou Mac OS X, o Python já vem instalado por padrão, restando instalar apenas o módulo. As instruções serão direcionadas para o sistema operacional Linux ou Mac OS X, no entanto, uma orientação será explicada para o Windows.

a) Baixar o arquivo no link: https://github.com/taironedias/iawork

b) Extrair o arquivo em uma pasta no computador.

c) Abrir o terminal e navegar até a pasta. Por exemplo, supondo que tenha baixado na pasta Documentos colocar a seguinte linha de comando:

d) (Para Linux):

e) $ cd Documentos/iawork/

f) (Para Mac OS X):

g) $ cd Documents/iawork

h) Depois chamar o interpretador do pyhton com o nome do arquivo “batman.py”. Ou seja,

i) $ python3 batman.py

j) Caso, o jogo não inicie, deve-se instalar o módulo.

k) $ python3 -m pip install -U pygame --user

l) Neste link (https://www.pygame.org/wiki/GettingStarted) apresenta instruções para Linux, Mac OS X e Windows.

m) Repetir o passo i)

Para Windows, deve-se baixar alguma IDE que venha o pacote do Python. Por exemplo, o Spyder ou PyCharm, são IDEs que já vem com todos os pacotes necessários para o desenvolvimento de um algoritmo em python. Segue o link de download, instalação e uso do Spider: https://youtu.be/tfQyw15m2I8

Com a IDE instalada e seguindo as orientações para a instalação do módulo do pygame no Windows, é só abrir o programa e a pasta onde está o arquivo batman.py e clicar no botão de executar. Mas, não há garantias de funcionamento, uma vez que esse trabalho foi realizado em um outro sistema. Dessa forma, segue o link de um *screencast* realizado sobre o jogo dias antes da entrega do trabalho, no sistema Mac OS X: https://youtu.be/_k6O84r-PuI

Para abrir o código, é necessário qualquer editor padrão de texto ou IDEs de desenvolvimento, como: Notepad++, Bloco de Notas, Gedit, Atom, entre outros. No Windows instalando o Spyder ou PyCharm, é possível visualizar o código antes ou após a execução do programa. O código inteiro não está comentado, porém trechos importantes ou de difícil entendimento do código estão.

No mais, a disciplina ajudou a aplicar conceitos já conhecidos na prática de desenvolvimento de software, porém sem fundamentos teóricos. Tendo assim um conhecimento apurado na área, uma organização e discernimento, onde dado um problema, qual a melhor técnica a ser utilizada, dentro de critérios estabelecidos como: tempo de processamento, desenvolvimento, memória disponível, entre outros.

REFERÊNCIAS

Inteligência Artificial em Jogos. Acessado em 23 de agosto de 2018. Disponível em <https://www.maxwell.vrac.puc-rio.br/7861/7861_3.PDF>

NASCIMENTO, J.; YONEYAMA, T. Inteligência artificial. **Editora Blucher**, 2000.

KISHIMOTO, André. Inteligência artificial em jogos eletrônicos. **Academic research about Artificial Intelligence for games**, 2004.

Artificial intelligence. Wikipedia. Acessado em 23 de agosto de 2018. Disponível em <https://en.wikipedia.org/wiki/Artificial_intelligence>

Get Started. Python. Acessado em 23 de agosto de 2018. Disponível em <https://www.python.org/about/gettingstarted/>

Getting Started. Pygame. Acessado em 23 de agosto de 2018. Disponível em <https://www.pygame.org/wiki/GettingStarted>