# Wumpus World

Zaharia Robert Jan

December 8, 2022

## 1 Introduction

### 1.1 Context

In cadrul acestui proiect se doreste implementarea unui Ai pentru jocul "Wumpus World". WUmpus World este un joc in care eroul trebuie sa parcurga un labirint pentru a ajunge la aur si a iesi nevatamat. Obstacolele intalnite in acest joc sunt monstrul Wumpus dar si gropile in care eroul nostru poate sa cada.

### 1.2 Specificatii

Pentru a crea acest interfata grafica acest joc vom folosi un framework realizat in python. Iar pentru realizarea miscarilor de catre agentul nostru vom folosi Prover9.

## 2 Realizarea programului

### 2.1 Rulare

Programul se ruleaza deschizand un terminal in folderul cu programul si ruland "python2 run.py". Apoi se apasa sageata dreapta pentru a derula miscarile agentului. Schimbarea hartii se poate realiza prin modificarea fisierului "run.py" unde se inlocuieste cu fisierul hartii dorite.

### 2.2 Partea de Python

In cadrul programului python se va realiza interfata grafica a jocului si se va asigura desfasurarea acestuia folosind functiile din python pentru a progresa. In fisierul "agent.py" agentul trimite informatii catre prover.py si apeleaza funtii din acesta pentru a verifica daca camera este goala. Metoda "make-move" incearca sa gaseasca camerele adiacente pozitiei curente care sunt goala. Daca nu reuseste sa gaseasca nicio camera goala atunci se va intoarce in camerele deja vizitate si va incerca sa gaseasca acolo o camera goala. Iar daca nici aici nu gaseste o camera goala atunci va trage in camera unde se afla wumpus daca ii stie locatia exacta altfel se va opri concluzionand ca nu exista o cale sigura pana la aur.

In fisierul "prover.py" se realizeaza modificarea fisierului prover metodele acestuia adaugand atribute in fisierul prover.

### 2.3 Partea de Prover9

In fisierul "wumpus.p9" avem codul verificat de prover 9 caruia ii vom adauga toate informatiile dobandite de-a lungul jocului. Fiecare camera avand coordonate (x,y) . Prima parte a codului verifica existenta wumpus-ului.

Primele 4 linii de cod din figura 2 verifica daca exista stench in camera. Urmatoarea linie verifica posibilitatea de a fi un Wumpus in camera verificata. Iar cealalta linie se asigura ca nu exista Wumpus in camera verificand daca lipseste un stench intr-una din camerele vecine.

In a doua jumatate a codului verificam daca exista o groapa in camera (figura 3). Urmatoarea linie verifica posibilitatea de a fi o groapa in camera verificata. Iar cealalta linie se asigura ca nu exista o groapa in camera verificand daca lipseste o briza intr-una din camerele vecine.

```
def make_move(self):
    self.inspect_position()
    print "Agent is checking adjacent hidden cells"
    cell = self.get_adjacent_safe_hidden_cell()
    if cell is not None:
        print "!Agent found safe hidden cell", cell
        return self.travel_to(cell)
    print "Agent could not find safe hidden cell"

    print "Agent is re-checking visited cells"
    for cell in reversed(self.visited[:-1]):
        print "Agent is checking", cell
        cell2 = self.get_adjacent_safe_hidden_cell(cell)
        if cell2 is not None:
            print "Agent found safe hidden cell", cell2
            return self.travel_to(cell2)

    if self.wumpus!=None:
        x,y=self.wumpus
        print("Agent Knows where wumpus is and fires at (%s,%s)",x,y)
        self.safe.add(self.wumpus)
        self.safe.add(x+1,y)
        self.safe.add(x-1,y)
        self.safe.add(x, y+1)
        self.safe.add(x, y-1)

    print"Agent is re-checking visited cells"
    for cell in reversed(self.visited[:-1]):
        print
        "Agent is checking", cell
        cell2 = self.get_adjacent_safe_hidden_cell(cell)
        if cell2 is not None:
            print
            "Agent found safe hidden cell", cell2
            return self.travel_to(cell2)
```

Figure 1: Make-move

```
chk_wumpus(x,y) & stench(x+1,y) -> stenchr(x+1,y).
chk_wumpus(x,y) & stench(x,y+1) -> stenchu(x,y+1).
chk_wumpus(x,y) & stench(x+-1,y) -> stenchl(x+-1,y).
chk_wumpus(x,y) & stench(x,y+-1) -> stenchd(x,y+-1).
chk_wumpus(x,y) & (stenchr(x+1,y) | stenchl(x+-1,y) | stenchu(x,y+1) | stenchd(x,y+-1)) -> maybe_wumpus(x,y).
chk_wumpus(x,y) & (-stench(x+1,y) | -stench(x+-1,y) | -stench(x,y+1) | -stench(x,y+-1)) ->no_wumpus(x,y).
```

Figure 2: Check Wumpus

```
breeze(x,y)->maybe_pit(x+1,y).
breeze(x,y)->maybe_pit(x+-1,y).
breeze(x,y)->maybe_pit(x,y+1).
breeze(x,y)->maybe_pit(x,y+-1).


chk_pit(x,y) & (-breeze(x+1,y) | -breeze(x+-1,y) | -breeze(x,y+1) | -breeze(x,y+-1)) ->no_pit(x,y).
chk_pit(x,y) & ( (breeze(x+1,y) & breeze(x+-1,y)) | (breeze(x,y+1) & breeze(x,y+-1)) | (breeze(x,y+1) & breeze(x+-1,y)) | (breeze(x,y+-1) & breeze(x+-1,y)) | (breeze(x,y+-1) & breeze(x+1,y)
```

Figure 3: Check Pits

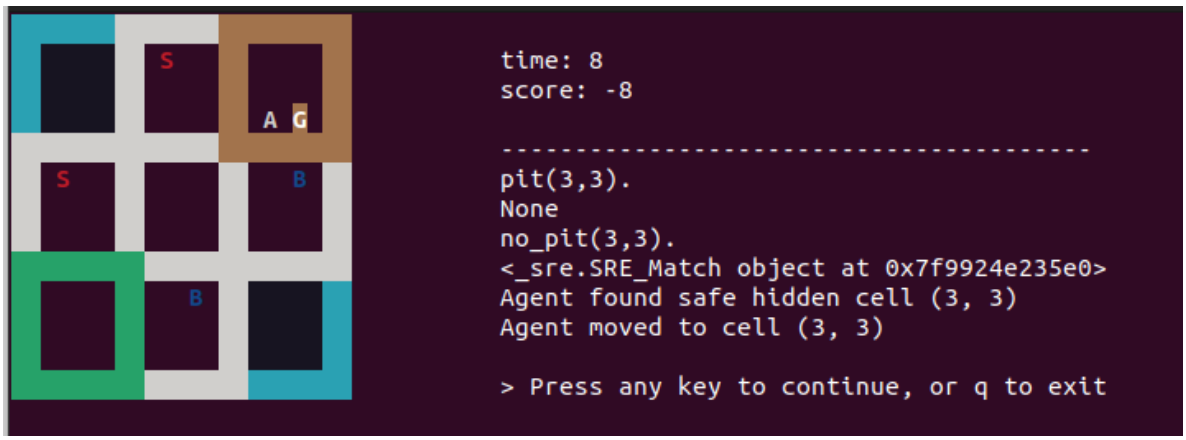


Figure 4: Harta initiala


Figure 5: Harta 5x5

## 3 Testare

Primul test realizat a fost pe o harta initiala de 3x3 unde agentul a trebuit sa realizeze ca camera de la coordonatele (2,2). In figura 4 se poate observa cum agentul nostru a gasit aurul la coordonatele (3,3). Dupa ce a gasit aurul agentul se intoarca la locatia initiala pe drumul pe care a venit (fig 4).

Al doilea test a fost realizat pe o harta initiala de 5x5 unde agentul a reusit sa gaseasca aurul la coordonatele (5,4)(figura 5).

Mapele pot fi create creand un fisier text unde se vor adauga informatiile regasite in fiecare camere (stench, wumpus,breeze,pit,gold) si coordonatele.

Teste ulterioare au dus la erori cand dimensiunea mapei depaseste 5x5. Erorile apar in prover9 in urma adaugarii prea multor atribute. In urma acestor erori programul nu poate distinge camerele sigure de cele cu wumpus sau cu groapa.