**Algorithm**

**Harrison Lara**

**Program 8: Wumpus World**

**Main File: Program8.py**

Handles all user interactions, gets input, checks it for validity, passes valid input to methods of WumpusWorld object, provides user feedback based on results, computes final score.

**Imported File 1: Map.py**

class OffMapError, class Cell, class Map should all be in source file Map.py

class OffMapError(Exception): raised if attempt to go off map.

Methods:

\_\_init\_\_(self)

class Cell(object): Models one cell (square) in the map.

Data:

self.hasWumpus 🡺 Boolean

self.hasGold 🡺 Boolean

self.hasPit 🡺 Boolean

self.hasBreeze 🡺 Boolean

self.hasStench 🡺 Boolean

Methods:

\_\_init\_\_(self)

class Map: manages the entire map, assigns values to cells, reports status of map cells

Data:

self.grid 🡺 5x5 (0,0) – (4,4)

Methods:

\_\_init\_\_(self)

onGrid(self, r, c) 🡺 Boolean

offGrid(self, r, c) 🡺 Boolean

reset(self) 🡺 User input

isBreezy(self, r, c) 🡺 Boolean

isSmelly(self, r, c) 🡺 Boolean

isGlinty(self, r, c) 🡺 Boolean

hasWumpus(self, r, c) 🡺 Boolean

hasPit(self, r, c) 🡺 Boolean

**Imported File 2: Wumpus.py**

class WumpusWorld: contains a map, records player movement & current game state

Data:

self.worldmap

self.WumpusAlive 🡺 Boolean

self.playerAlive 🡺 Boolean

self.playerHasGold 🡺 Boolean

self.playerHasArrow 🡺 Boolean

self.playerMoves 🡺 Boolean

self.playerRow 🡺 (x axis)

self.playerCol🡺 (y axis)

Methods:

\_\_init\_\_(self)

Most of these use current player location

stepEast(self) 🡺 User Input East

stepWest(self) 🡺 User Input West

stepSouth(self) 🡺 User Input South

stepNorth(self) 🡺 User Input North

grab(self, r, c) 🡺 User Input Grab

grabGold(self) 🡺 User Input Grab Gold

# checks current r, c, calls

fire(self, direction) 🡺 User Input Fire

canClimb(self) 🡺 User Input Climb

feelBreeze(self) 🡺 Boolean

smellStench(self) 🡺Boolean

seeGlint(self) 🡺Boolean

hasWumpus(self) 🡺 Boolean

hasPit(self) 🡺Boolean

5x5 grid, with cells numbered from 0,0 to 4,4.

The player starts at 0,0.

Players may take the following actions:

North, South, East, West (Move one square in that direction)

5 pits, 1 Wumpus, 1 gold (all random placement, but stationary)

Pits > gold,

Wompus > gold (unless shot)

Wompus > pits

Climb out is at 0,0

Only the player can be placed at 0,0

**Scoring:**

+1000 Climbed out with gold

+100 Climbed out without gold

-10 Fired weapon (only one arrow)

-1 per move

Score 0 if the player died

• Write a Cell class. This class will model one square in the map. A Cell simply has a few boolean variables; for example, hasPit, hasGold, isBreezy, etc.

• Write a Map class. A Map class contains a 2-dimensional list of Cells. It also has the following methods:

◦ isBreezy(self, row, col): returns True if the specified cell is breezy, False otherwise.

◦ isSmelly(self, row, col): returns True if the specified cell has a stench, False otherwise

◦ isGlinty(self, row, col): returns True if the gold is in that square, false otherwise.

◦ hasWumpus(self, row, col): returns True if the Wumpus is there, false otherwise.

◦ hasPit(self, row, col): returns True if the cell has a pit, False otherwise

**Extra Notes:**

## bottom left corner will be 0,0, player spawns there to start

## the grid will be 25 cells (5x5) or 0-4 on the x and y axis

## Wumpus and gold spawns on one cell and doesn’t move

## 5 of the cells will be pits

## Gold can be in a pit, never reachable gold

## pit and Wumpus can be in the same cell but the Wumpus is greater than the pit

## adjacent to Wumpus - smell stench

## same cell as the gold - see a glint (grab gold)

## user tries to go off grid- player feels a bump

## player has one arrow to shoot and will go across all the cells in that direction

## score is always zero if you die

## minus score for shooting arrow and each move is a point deduction

## use classes and functions given in the pdf

## self.haswompus, etc should all be under the init(self) and Booleans

## return true if on grid, off grid will return false

## just write the code for each class under class given

## no need to make any more classes

## can use an int or a string with direction to go a certain way

## should have 2 outsource files with the other classes

## import the two other files

## output that it is dark each time unless other

## greet user and tell them their commands

## climb out is at 0,0

## import (name of file)