Hunt the Wumpus

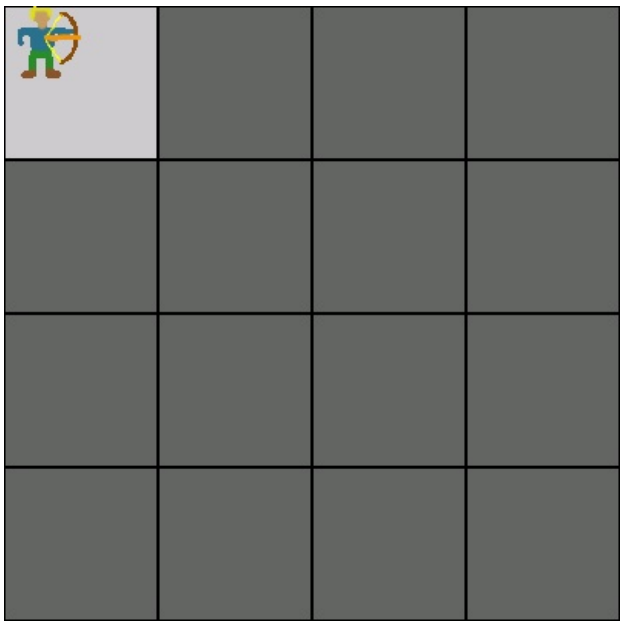
Objectives

- Use lists
- Use loops
- Use conditionals
- Define and use functions
- Create and use code modules

Assignment

“Hunt the Wumpus” is a game that has been around in computing for over 40 years. Curiously, the objective in the game is not to kill the wumpus, but to find the gold. What’s a wumpus you say? The wumpus is a smelly, scary monster lurking in a dark cavern.

Below is a picture of the start of the game. The adventurer is standing in the first cell in the cavern, and doesn’t know where the wumpus or the gold are.



If you stand in the same cell as the wumpus, it will eat you! The picture below shows the game after the adventurer has explored several cells, and unfortunately stepped into the cell where the wumpus is sitting. The wumpus likes a little ketchup on his adventurers. Game over.

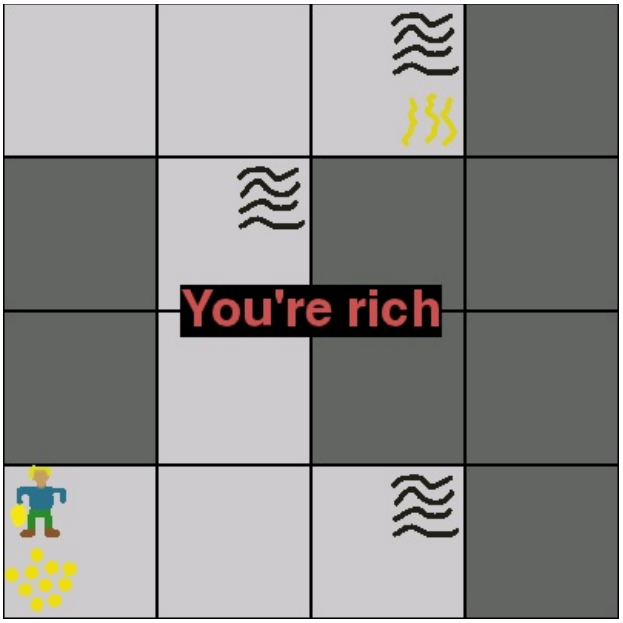
Luckily, the wumpus’ stench warns you when it is nearby, and you can stay away from it. Notice the wavy vertical yellow stench lines in the picture.

Also, you can fall into pits contained in the cavern and be killed!

Luckily, if you are near a pit, you will feel a small breeze. Again, warning you that danger is nearby. Notice the wavy horizontal black lines in the picture.

The problem is, you don’t know which direction the wumpus is when you smell the stench, and you don’t know which direction the pit is when you feel a breeze. In fact, because there may be more than one pit, you don’t know if there are pits in many directions. As you explore the cavern, you will be able to see the clues contained in more and more cells. If you explore the cell containing the gold, before you explore a cell containing the wumpus or a pit, then you grab the gold and win.

Notice the adventurer holding the gold below. Can you tell where the wumpus is? Can you identify any cells that must contain a pit? Can you identify any cells that must be safe (no pit and no wumpus)? Can you identify any cells that may contain a pit but may be safe?



If you explore a cell with the wumpus or a pit before finding the gold, you die. If the gold happens to be contained in a cell with a pit or the wumpus, you will die before you get to pick it up.

In this assignment you will implement the majority of a hunt the wumpus game, following the design described here. The rest of the implementation will be given to you. The game will be built using four separate modules: `wumpus_main`, `wumpus_draw`, `wumpus_logic`, and `wumpus_data`. At the end of this document there is a glossary of terms, and a reference description of each function in each module. In addition to the four modules you will implement, the pygame helper module `game_mouse.py` will be used to display the game and to interact with the user.

You will use a bottom-up implementation strategy for this assignment. You will first implement the `wumpus_data` module, then the `wumpus_logic` module and finally the `wumpus_draw` module. The `wumpus_main` module will be given to you, or you can implement it yourself, if you want the extra challenge. The instructions for implementing each of the modules follow the section on the starter kit.

Running Hunt the Wumpus from the Starter Kit

Download the starter kit:

- Python 2.7.x wumpus-starter-kit-2-7.zip

Unzip the files into a folder. We will call this folder your “example folder”. Open `wumpus_temp.py` in your Python editor (IDLE). Run the file. Play the game several times. Understanding what the game does will help you implement it. You should also gain pleasure from the exquisite drawings used in the display. How often do you find the gold without dieing?

To quit the game, click the “X” on the window title bar, or press the `ESC` key.

Implementing `wumpus_data` (Part 1)

In our version of hunt the wumpus, all of the game information is stored in one big list. For example, the list stores the current location of the adventurer, the location of the gold and the location of the wumpus. It also tracks information like all of the cells that have been visited by the adventurer. For a complete description of the world data list, see the section titled “The World Data List” at the end of this file.

In this part, you will implement all of the functions in the `wumpus_data` module. A list of the functions, and their descriptions can be found at the end of this document, in the section titled “Functions in the `wumpus_data` module”.

We recommend that you create a new folder, unzip the starter kit into this folder, and use this as your “working folder.” Create a new file `wumpus_data.py` here and implement all of the functions for this module. When you have implemented them correctly, you will be able to run the program here.

Here’s a little code and comments to get you started:

```
import random
```

```
def setDimensions(data, width, height):
    data[0] = width
    data[1] = height

def getDimensions(data):
    return (data[0], data[1])
# Implement setCellSize
# Implement getCellSize
# Implement setPits
# Implement getPits
# Implement setVisible
# Implement getVisible
# Implement setWumpusPosition
# Implement getWumpusPosition
# Implement setGoldPosition
# Implement getGoldPosition
# Implement setHaveGold
# Implement getHaveGold
# Implement setHaveArrow
# Implement getHaveArrow
# Implement setIsAlive
# Implement getIsAlive
# Implement setAdventurerPosition
# Implement getAdventurerPosition

# You must add to initializeData so that it will create
# correct values for the wumpus world
# The code provided here makes place holders for the values
def initializeData(width, height, cell_size, debug=0):
    pits    = []
    visible = []
    for i in range(width*height):
        pits.append(False)
        visible.append(False)
    data = [ width, height, cell_size,
             pits, visible,
             width-1, height-1,
             1, 1,
             False, True, True,
             0, 0,
             ]
    return data
```

Most of the functions in this module are straight-forward. The only involved one is `initializeData`. Read the description of the world data list and this function, and pay attention in class.

You should write test routines to check your module. If you want to do additional tests on your module, use the `wumpus_data_black_box.py`.

Save a copy of this working `wumpus_data.py` where you cannot accidentally delete or modify it. You are ready to pass off and submit `wumpus_data.py`.

Implementing `wumpus_logic` (Part 2)

The `wumpus_logic` module makes all of the important decisions in the wumpus game and advances the state of the game. For example, the `handleMouseClick` function processes a mouse click, and based on the state of the game moves the adventurer and marks the new cell visited.

In this part, you will implement all of the functions in the `wumpus_logic` module. A list of the functions, and their descriptions can be found at the end of this document in the section titled “Functions in the `wumpus_logic` module”.

Create a new file `wumpus_logic.py` in your working folder and implement all of the functions for this module. When you have implemented them correctly, you will be able to run the program here.

Here’s a little code and some comments to get you started:

```
from wumpus_data import *

# Implement cellContainsWumpus
# Implement cellContainsGold
```

```
def cellContainsPit(data, x, y):
    (width, height) = getDimensions(data)
    pits = getPits(data)
    i = y * width + x
    return pits[i]

# Implement cellIsVisible
# Implement cellIsInCavern

def neighborCellIsVisible(data, x, y):
    if cellIsInCavern(data, x + 1, y) and cellIsVisible(data, x + 1, y):
        return True
    elif cellIsInCavern(data, x - 1, y) and cellIsVisible(data, x - 1, y):
        return True
    elif cellIsInCavern(data, x, y + 1) and cellIsVisible(data, x, y + 1):
        return True
    elif cellIsInCavern(data, x, y - 1) and cellIsVisible(data, x, y - 1):
        return True

    return False

# Implement neighborCellContainsPit
# Implement neighborCellContainsWumpus
# Implement handleMouseClick
# Implement setCellVisible
# Implement visitCell
```

Read the function descriptions carefully, and pay attention in class.

You should write test routines to check your module. If you want to do additional tests on your module, use the `wumpus_logic_black_box.py`.

Save a copy of this working `wumpus_logic.py` where you cannot accidentally delete or modify it. You are ready to pass off and submit `wumpus_logic.py`.

Implementing `wumpus_draw` (Part 3)

In this part, you will implement all of the functions in the `wumpus_draw` module. A list of the functions, and their descriptions can be found at the end of this document in the section titled “Functions in the `wumpus_draw` module”.

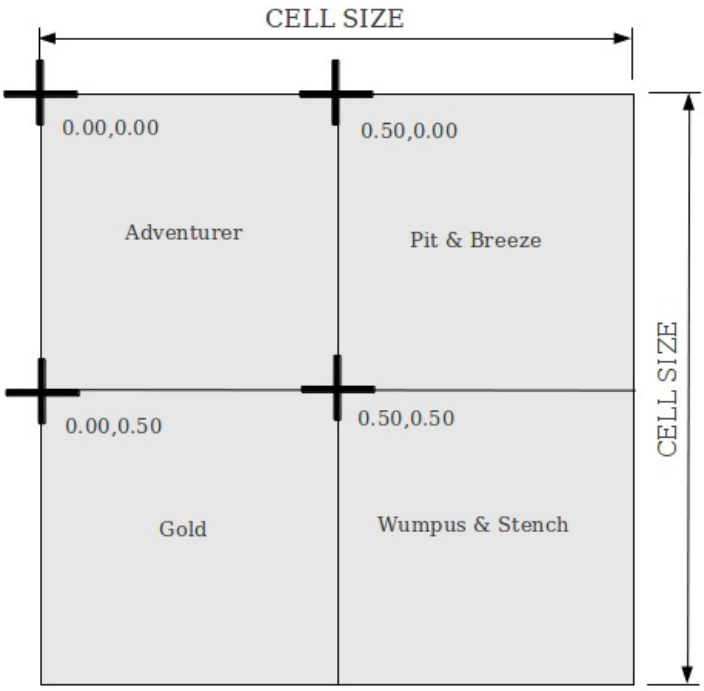
Create a new file `wumpus_draw.py` in your working folder and implement all of the functions for this module. When you have implemented them correctly, you will be able to run the program here.

Here’s a little code and comments to get you started:

```
import pygame
from wumpus_data  import *
from wumpus_logic import *

def loadImages():
    global HIDDEN, VISIBLE, BACKGROUND, MESSAGE_COLOR
    global WUMPUS_PIXMAP, PIT_PIXMAP, GOLD_PIXMAP, STENCH_PIXMAP, BREEZE_PIXMAP
    global ADVENTURER_GOLD_PIXMAP, ADVENTURER_DEAD_PIXMAP
    global  ADVENTURER_ARROW_PIXMAP, ADVENTURER_NO_ARROW_PIXMAP

    # Set Cell Colors
    # Load Images

def drawPit(surface, data, x, y):
    pixmap    = PIT_PIXMAP
    dx, dy    = .5, .0
    cell_size = getCellSize(data)
    exists    = cellContainsPit(data, x, y)
    point     = ((x+dx)*cell_size, (y+dy)*cell_size)
    if exists:
        surface.blit(pixmap, point)

# Implement drawWumpus
# Implement drawGold
# Implement drawBreeze
# Implement drawStench
```

```
# Implement drawCell
# Implement drawAdventurer
# Implement drawGameOver
# Implement updateDisplay
```

Each cell is drawn within a square that is 'cell size' pixels by 'cell size' pixels. The pit and breeze are both drawn in the upper right hand quadrant of the square. That means the upper left corner of the image is 0.50 of a cell from right and 0.00 of a cell down from the top left hand corner of the square. The wumpus and the stench are drawn in the bottom right hand quadrant of the square. The gold is drawn in the bottom left hand quadrant of the square. The adventurer is drawn in the upper left hand quadrant of the square. See the diagram.



Save a copy of this working `wumpus_draw.py` where you can not accidentally delete or modify it. You are ready to pass off and submit `wumpus_draw.py` and move on to the next module.

Implementing `wumpus_main` (Not required)

The `wumpus_main` module has already been created and provided for you. This is the part of the program that will need to change if you would like to add different kinds of user input, such as right mouse button click to shoot the arrow.

Debug Mode

Every time you run the game, the pits, the wumpus and the gold are randomly placed on the board. This makes the game more fun, but makes it more difficult to debug as you write the program.

If you correctly implemented the debug mode into the `wumpus_data` it will always place the pieces in the same cells. In `wumpus_temp.py`, where the `debug` variable is assigned, change the value to `1` for fixed items. If you change the value to `2`, then you will see the map, but all cells will be visible from the start of the game. This will be very useful as you complete the `wumpus_draw` module. Remember to change it back to `0` when you are done debugging.

Conclusion

Congratulations! You have written a classic game that will entertain your friends and family for hours.

Optional challenges

If you think you can improve on the stunning graphics, then create GIF image files, and replace the images

that came with the starter kit. Unless you change the cell size, the images should be 50 pixels by 50 pixels.

In the full game, the adventurer can use the arrow to shoot and kill the wumpus. This will sometimes allow the adventurer to explore more of the map, and find the gold. Add this functionality to your program.

Glossary of terms

- Adventurer: The player that is exploring the cavern.
- Cavern: The entire map of area to be explored by the adventurer.
- Cell: A single location in the cavern.
- Wumpus: A smelly monster that lives in the cavern. If the adventurer and the wumpus are in the same cell, the wumpus will eat the adventurer. There is always exactly 1 wumpus in the cavern.
- Gold: A valuable treasure the adventurer is seeking. If the adventurer and the gold are in the same cell, the adventurer will pick up the gold. There is always exactly 1 gold in the cavern.
- Pit: A deep hole in the floor of the cavern that consumes all of the floor space of a cell. If the adventurer enters a cell that contains a pit, the adventurer will fall into the pit and die. Each cell has a 20% chance of containing a pit.
- Stench: The odor from the wumpus. The stench can be detected in any cell next to the one that contains the wumpus.
- Breeze: Heat convection from the deep pits cause a small breeze to blow in cells that neighbor a cell that contains a pit.
- Width: The number of cells from side to side in the cavern.
- Height: The number of cells from top to bottom in the cavern.
- Cell Size: The number of pixels on each side of the displayed cells.
- Visible: A property of a cell. If the adventurer has ever been in a cell, then it is visible.

Functions in the `wumpus_main` module

- `main` takes no parameters and does not return anything. This function calls `initializeGraphics` and `initializeData`, passing in the `width`, `height` and `size` arguments. While the adventurer is alive and the adventurer does not have the gold, the display is updated with `updateDisplay`, the user selects an unvisited cell through `clickCell`, and the selected cell is visited with `visitCell`. After the adventurer dies or finds the gold, the display is updated one last time, then the program waits for a final user click with `getMouse`, then closes the window.

Functions in the `wumpus_draw` module

- `loadImages` takes no parameters. It returns nothing. This function creates colors for drawing, and loads images from files for the wumpus, pits, gold, stench, breeze, and various states of the adventurer. The colors and images are saved in global variables to be used by other drawing functions.
- `drawPit` takes 4 parameters, the graphics surface, the wumpus world data list, the x and the y coordinates of the cell to draw the pit in. It does not return anything. This function checks if the cell contains a pit. If it does contain a pit, then the location of the pit's image in the window is calculated, and the image is drawn in the window.
- `drawWumpus` takes 4 parameters, the graphics surface, the wumpus world data list, the x and the y coordinates of the cell to draw the wumpus in. It does not return anything. This function checks if the cell contains the wumpus. If it does contain the wumpus, then the location of the wumpus's image in the window is calculated, and the image is drawn in the window.
- `drawGold` takes 4 parameters, the graphics surface, the wumpus world data list, the x and the y coordinates of the cell to draw the gold in. It does not return anything. This function checks if the cell contains the gold. If it does contain the gold, then the location of the gold's image in the window is calculated, and the image is drawn in the window.
- `drawBreeze` takes 4 parameters, the graphics surface, the wumpus world data list, the x and the y coordinates of the cell to draw the breeze in. It does not return anything. This function checks if the cell

has any neighbor that contains a pit. If a neighbor does contain a pit, then the location of the breeze's image in the window is calculated, and the image is drawn in the window.

- `drawStench` takes 4 parameters, the graphics surface, the wumpus world data list, the x and the y coordinates of the cell to draw the stench in. It does not return anything. This function checks if the cell has a neighbor that contains the wumpus. If a neighbor does contain the wumpus, then the location of the stench's image in the window is calculated, and the image is drawn in the window.

- `drawCell` takes 4 parameters, the graphics surface, the wumpus world data list, the x and the y coordinates of the cell to draw. It does not return anything. This function calculates the location of the cell in the window. If the cell is not visible, then a rectangle is drawn with the hidden background color. If the cell is visible, then a rectangle is drawn with the visible background color, and the functions `drawPit`, `drawWumpus`, `drawGold`, `drawBreeze`, and `drawStench` are called, in that order. Finally, the function draws a narrow border around the cell.

- `drawAdventurer` takes 2 parameters, the graphics surface, and the wumpus world data list. It does not return anything. This function calculates the location of the adventurer's image in the window and draws the adventurer's image in the window. If the adventurer is dead, the dead adventurer image is used. Otherwise, if the adventurer has the gold, the adventurer with gold image is used. Otherwise, if the adventurer is still holding the arrow, the adventurer with arrow image is used. Otherwise, the adventurer without arrow image is used.

- `drawGameOver` takes 2 parameters, the graphics surface, and the wumpus world data list. It does not return anything. If the adventurer is dead, the function displays a farewell message. If the adventurer is holding the gold, the function displays a congratulation message. If the adventurer is neither dead nor holding the gold, the function does not display any message.

- `updateDisplay` takes 2 parameters, the graphics surface, and the wumpus world data list. It does not return anything. This function loops over all cells in the cavern, and calls `drawCell` for each one. It then calls `drawAdventurer` and `drawGameOver`. HINT: `drawCell` will be called once for each row/column intersection (cell).

Functions in the `wumpus_logic` module

- `cellContainsWumpus` takes 3 parameters, the wumpus world data list, the x and the y coordinates of a cell. It returns `True` if the wumpus is in the cell, and `False` if it is not in the cell. It compares the position of the wumpus with the coordinates of the cell passed in. The position of the wumpus is obtained with the `getWumpusPosition` function.

- `cellContainsGold` takes 3 parameters, the wumpus world data list, the x and the y coordinates of a cell. It returns `True` if the gold is in the cell, and `False` if it is not in the cell. It compares the position of the gold with the coordinates of the cell passed in. The position of the gold is obtained with the `getGoldPosition` function.

- `cellContainsPit` takes 3 parameters, the wumpus world data list, the x and the y coordinates of a cell. It returns `True` if there is a pit in the cell, and `False` if there is not a pit in the cell. It gets the list of pit values with the `getPits` function. It then calculates the index of the cell in the pits list using the cell's coordinates and the world's width. The pit value for the cell is returned.

- `cellIsVisible` takes 3 parameters, the wumpus world data list, the x and the y coordinates of a cell. It returns `True` if the cell is visible, and `False` if the cell is not visible. It gets the list of visible values with the getVisible function. It then calculates the index of the cell in the visible list using the cell's coordinates and the world's width. The visible value for the cell is returned.

- `cellIsInCavern` takes 3 parameters, the wumpus world data list, the x and the y coordinates of a cell. It returns `True` if the coordinates are legal, and `False` if they are not. It checks to see that the x and y coordinates are both greater than or equal to 0, and that x is less than the cavern's width, and that y is less than the cavern's height.

- `neighborCellIsVisible` takes 3 parameters, the wumpus world data list, the x and the y coordinates of a cell. It returns `True` if one of the direct neighbors to the cell is visible. Otherwise, it returns `False`. This function checks if any of the cells to the north, south, east or west of the cell are in the cavern, and is visible. Uses the `cellIsInCavern` and `cellIsVisible` functions.

- `neighborCellContainsPit` takes 3 parameters, the wumpus world data list, the x and the y coordinates of a cell. It returns `True` if one of the direct neighbors to the cell contains a pit. Otherwise, it returns `False`. This function checks if any of the cells to the north, south, east or west of the cell are in the cavern, and contains a pit. Uses the `cellIsInCavern` and `cellContainsPit` functions.

- `neighborCellContainsWumpus` takes 3 parameters, the wumpus world data list, the x and the y coordinates of a cell. It returns `True` if one of the direct neighbors to the cell contains the wumpus. Otherwise, it returns `False`. This function checks if any of the cells to the north, south, east or west of the cell are in the cavern, and contains the wumpus. Uses the `cellIsInCavern` and `cellContainsWumpus` functions.

- `setCellVisible` takes 3 parameters, the wumpus world data list, and the x and y coordinates of the cell to set as visible. It does not return anything. This function uses the width of the world, and the y and x coordinates to calculate the index of the cell in the visible list. It fetches the visible list from the wumpus world data list with the `getVisible` function, and sets the visible state for this cell to `True`.

- `visitCell` takes 3 parameters, the wumpus world data list, and the x and y coordinates of the cell to be visited. It does not return anything. This function checks if the cell being visited contains a pit or the wumpus using `cellContainsWumpus` and `cellContainsPit`. If so, the adventurer's `isAlive` state is set to `False` with the `setIsAlive` function. Otherwise, if the adventurer is still alive and if the cell being visited contains the gold (use `cellContainsGold`), the adventurer's `haveGold` state is set to `True` with the `setHaveGold` function. Finally the cell's visible state is set to `True` with the `setCellVisible` function.

- `handleMouseClick` takes 3 parameters, the wumpus world data list, the x and y pixel values of a mouse click. It does not return anything. This function calculates the cell that was clicked, using the cell size and the pixel location of the mouse click. If the adventurer is alive, and the adventurer does not have the gold, and the cell clicked has at least one visible neighbor, then the adventurer is moved to the clicked cell, and the clicked cell is visited.

Functions in the `wumpus_data` module

For each of these functions, refer to the positions of data items shown in "The World Data List" section below.

- `setDimensions` takes 3 parameters, the wumpus world data list, the width and the height of the cavern. It updates the values stored in the world data list with the values from parameters.

- `getDimensions` takes 1 parameter, the wumpus world data list. It returns the width and height of the cavern, from the world data list.

- `setCellSize` takes 2 parameters, the wumpus world data list and the new cell size. It updates the value stored in the world data list with the value from the parameter.

- `getCellSize` takes 1 parameter, the wumpus world data list. It returns the cell size, from the world data list.

- `setPits` takes 2 parameters, the wumpus world data list and the list of pit values for all cells in the cavern. It updates the pits list stored in the world data list with the list from the parameter.

- `getPits` takes 1 parameter, the wumpus world data list. It returns the list of pit values for all cells in the cavern, from the world data list.

- `setVisible` takes 2 parameters, the wumpus world data list and the list of visible values for all cells in the cavern. It updates the visible list stored in the world data list with the list from the parameter.

- `getVisible` takes 1 parameter, the wumpus world data list. It returns the list of visible values for all cells in the cavern, from the world data list.

- `setWumpusPosition` takes 3 parameters, the wumpus world data list, the x and y coordinates of the cell that contains the wumpus. It updates the values stored in the world data list with the values from parameters.

- `getWumpusPosition` takes 1 parameter, the wumpus world data list. It returns the x and y coordinates of the cell that contains the wumpus, from the world data list.

- `setGoldPosition` takes 3 parameters, the wumpus world data list, the x and y coordinates of the cell that contains the gold. It updates the values stored in the world data list with the values from parameters.

- `getGoldPosition` takes 1 parameter, the wumpus world data list. It returns the x and y coordinates of the cell that contains the gold, from the world data list.

- `setHaveGold` takes 2 parameters, the wumpus world data list, and a `True` or `False` value. It updates the data list to contain the new value. It does not return anything.

- `getHaveGold` takes 1 parameter, the wumpus world data list. It returns `True` if the adventurer is holding the gold, and `False` if the adventurer is not holding the gold.

- `setHaveArrow` takes 2 parameters, the wumpus world data list, and a `True` or `False` value. It updates the data list to contain the new value. It does not return anything.
- `getHaveArrow` takes 1 parameter, the wumpus world data list. It returns `True` if the adventurer is holding the arrow, and `False` if the adventurer is not holding the arrow.
- `setIsAlive` takes 2 parameters, the wumpus world data list, and a `True` or `False` value. It updates the data list to contain the new value. It does not return anything.
- `getIsAlive` takes 1 parameter, the wumpus world data list. It returns `True` if the adventurer is alive, and `False` if the adventurer is dead.
- `setAdventurerPosition` takes 3 parameters, the wumpus world data list, the x and y coordinates of the cell that contains the adventurer. It updates the values stored in the world data list with the values from parameters.
- `getAdventurerPosition` takes 1 parameter, the wumpus world data list. It returns the x and y coordinates of the cell that contains the adventurer, from the world data list.
- `initializeData` takes 4 parameters, the cavern width, the cavern height, the cell size, and the debug mode. It returns a list that contains all of the wumpus world data. This function randomly constructs a cavern with each cell having a 20% chance of containing a pit. The gold is randomly placed in one of the cells. The wumpus is also randomly placed in one of the cells. The adventurer's starting cell, which has the coordinates `(0, 0)`, can not contain a pit, the gold, or the wumpus. The adventurer always starts without the gold, with the arrow, and alive. The debug mode should be 0 for random boards, 1 for a fixed board, and 2 for the fixed board with all cells visible.

The World Data List

The wumpus world data list contains the following items with the index numbers indicated:

- [0],[1] = width, height : the dimensions of the world, in cells.
- [2] = cell_size : the number of pixels on the side of each cell in the display.
- [3] = pits : an array with one element per cell. The value of each element, pits[i] is True if there is a pit in the cell, and False if there is not a pit in the cell.

  indexed by i = y * width + x
- [4] = visible : an array with one element per cell. The value of each element, visible[i] is True if the adventurer has visited the cell, and False if the adventurer has not visited the cell.

  indexed by i = y * width + x
- [5],[6] = wumpus_x, wumpus_y : the coordinates of the cell that contains the wumpus.
- [7],[8] = gold_x, gold_y : the coordinates of the cell that contains the gold.
- [9] = have_gold : True if the adventurer is holding the gold, otherwise it is False.
- [10] = have_arrow : True if the adventurer is still holding the arrow, otherwise it is False.
- [11] = is_alive : True if the adventurer has not fallen in a pit and has not been eaten by the wumpus. Otherwise, it is False.
- [12],[13] = adventurer_x, adventurer_y : the location of the adventurer on the grid