Part 1: Mortal Kombat [5 Points]

In the world of Mortal Kombat, the ultimate battle between Scorpion and Sub-Zero is about to commence in the Earth realm. The air is thick with tension as these two legendary warriors prepare to face off in a fight to the death. Both fighters are masters of their respective arts, each with a full health bar, ready to unleash their fury upon one another. In this brutal, turn-based clash, the warriors will alternately strike at their opponent and ensure victory.

As the battle unfolds, every turn counts. Will it be Scorpion, with his fiery vengeance, who claims victory? Or will Sub-Zero's icy precision lead him to triumph? The outcome depends on the initial setup: who strikes first, and how powerful their moves can be.

Your task is to simulate this epic showdown given the initial conditions – who starts the battle and determines who emerges as the victor. Prepare yourselves, for this is Mortal Kombat, where only one warrior will be left standing. **Round 1, Fight!**

Game Description & Rules:
- **Players:** Scorpion and Sub-Zero.
- **Turn-Based Gameplay:** Players take turns to attack.
- **Turns:** The player specified by the first input number will take the first turn in the first round. The player who didn't take the first turn in the previous round will start in subsequent rounds.
- Assume the **branching factor = 2** and the **max depth of the game tree = 5**
- **You have to create the appropriate number of leaf nodes and assign them utility values (-1 if scorpion wins, and 1 if sub-zero wins)**

Input

- One single-digit number. The number indicates which player starts first (0 for Scorpion, 1 for Sub-Zero).

Output

- The name of the game-winner.
- The number of rounds played.
- The winner of each round.

Sample Input:

```
0
```

Sample Output:

```
Game Winner: Scorpion

Total Rounds Played: 3

Winner of Round 1: Sub-Zero

Winner of Round 2: Scorpion

Winner of Round 3: Scorpion
```

Now apply the alpha-beta pruning algorithm to simulate the problem and find the winner.

Part 2: Games with Magic [5 Points]

In a strategic game simulation, Pacman and a ghost engage in a sequence of moves across three levels of depth. The competition starts with Pacman aiming to reach a power pellet by making the first move. The ghost then attempts to thwart Pacman's progress at the second level, followed by Pacman's final moves to reach the pellet at the leaf nodes, where the outcomes are scored as **3, 6, 2, 3, 7, 1, 2, 0**. An added strategic element is Pacman's ability to wield **dark magic**, allowing him to control the ghost's move at a specific cost, **c**. This cost is deducted from his score, influencing his overall strategy to reach the pellet.

You are tasked with developing the function **pacman_game(int c)** to determine the most effective strategy for Pacman, considering whether the use of dark magic, given its cost, is beneficial.
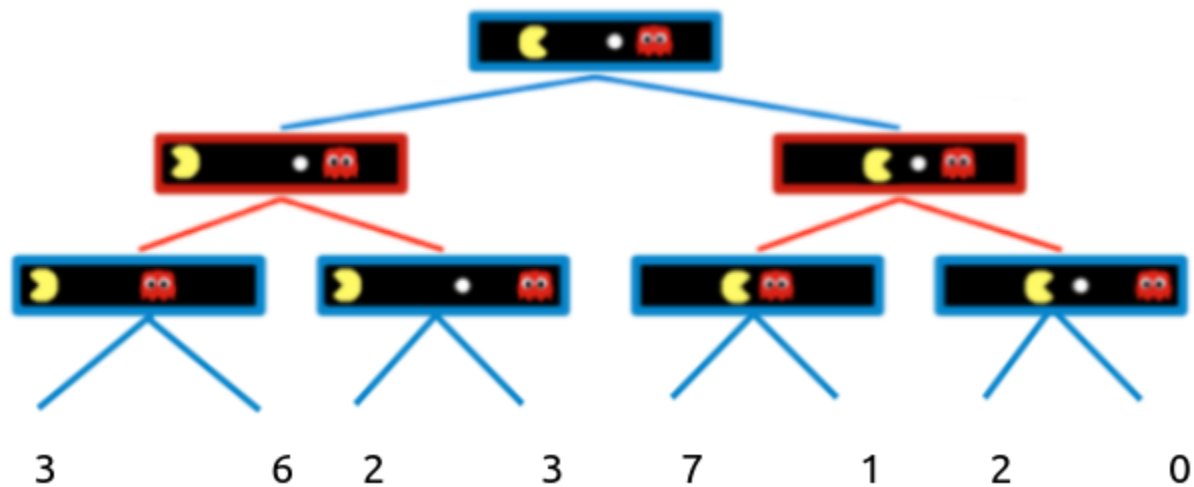
Input (c)	Output
2	The new minimax value is **5**. Pacman goes right and uses dark magic
5	The minimax value is **3**. Pacman does not use dark magic

In sample output **1**, the left subtree has the highest value 6 and the right subtree has the highest value **7**. If Pac-Man **moves to the left and uses dark magic** the result will be **6 - 2 = 4**, If Pac-Man **moves to the right and uses dark magic** the result will be **7 - 2 = 5**.

And now, **using Alpha-beta pruning, find** the final value of the root node **without using dark magic**. Then check whether using dark magic is advantageous for Pac-Man or not.

[Hint: The final value of the root node will be 3. So using dark magic is advantageous for Pac-Man in both directions, but it's more beneficial when Pac-Man goes right.]

Part 3: Food for thought [0 Points]

- **Is the first player always a maximizer node?**
- **Can alpha-beta pruning handle stochastic environments?**