

INTRODUCTION

This paper presents how the game called Super-Nim was implemented in python programming language. The methods and techniques applied for the design and implementation of the game are mentioned in this paper. The game can be played in three different ways: 1) game between random and random player, 2) game between random and AI player and 3) game between human and human player. The user can select the game option. There is a game manager program that runs the game between two players. The game can be between two random players or between random player and AI player or between two human players.

GAME STRUCTURE AND ARCHITECTURE

The “max” and “min” are used to indicate two players. In case of a game between random players, max and min both are random players. In case of a game between random and AI player, min is a random player whereas max is the AI player. Similarly, max and min are two human players in case of a game between two human players. All the necessary properties to define the state of a game is stored in a tuple which is a sequence of Python objects like a list. But unlike a list, objects of tuples cannot be altered. These properties include player whose turn it is, board, a score of max and score of min.

For example:

```
state = ("min", 2, 2, 4, 2, 0, 0)
```

In above state,

Number of elements in the tuple or length of tuple = 7

The first element “min” would mean that it is the turn of min.

The last two elements indicate the score of max (second last element) and score of min (last element). In above state, a score of both of them is 0.

The remaining elements in the tuple denote the board. In above state, the board tuple is (2,2,4,2) which further means the board contains a heap with 2 sticks, second heap with 2 sticks, a heap with 4 sticks and third heap with 2 sticks.

The way that random player, AI player and the human player plays the game is coded in different ways.

Random player: The random player uses succ(state) method of the class Super_Nim to generate all possible moves from the provided state. Then, the random player uses random_move(list of possible moves) which has the list of the possible states or moves as a parameter and returns a legal state by the random selection.

The random selection is done by using the random class in python. `Random.choice(seq)` is used which returns a random element from the sequence 'seq'. Here, sequence 'seq' is the list of all possible states or moves.

AI player: The AI player also uses `succ(state)` method to generate all possible moves from the provided state. Then, the AI player uses `ai_move(list of possible moves)` method which has the list of the possible states as a parameter and returns a state or move that has more probability of winning the game compared to other moves.

Human player: The human player also generates all possible moves from the provided state using `succ(state)` method. Then, the human player uses `human_move(list of possible moves)` method which has the list of the possible states as a parameter and returns a state or move chosen by the human player according to his/her will.

MOVE GENERATION

In the program, `succ(state)` method generates all possible moves or states from the state or move which is provided to the method as an argument. If the argument state has "max", then all possible moves have "min" as their first element. If the argument state has "min", then all possible moves have "max" as their first element. That part was achieved in the program by the use of the if-elif conditional statement. The principles of the game were followed for move generation. For placing one heap to another, for loop is used in the program for the iteration of elements in board tuple so that every heap gets added to another heap. Also, the addition of heap to the same heap and the addition of two heaps with equal sticks are avoided by checking the condition before addition. Capping to 10 is applied if the sum of a stick of two heaps is greater than 10. After the possible moves formed by adding two heaps are generated, they are stored in the list that has been made to store all possible moves. Then, another case is considered for move generation, i.e. cash-in if the heap has 2 sticks and split of the heap with an even number of sticks other than 2. All the possible moves formed considering that are also stored in the previous list of possible states. If there is a cash-in, the score of the player also changes. After that, Ulam step is applied for move generation. If the heap has no even number of sticks, then Ulam step is applied, and possible results are stored in the same previous list. So, the list contains all possible moves that can be played by a player from the provided state.

ALGORITHM USED BY AI PLAYER

The random player randomly selects the move from possible moves, and the human player selects the move according to its free will. But, the case with AI player is not similar since it has to make a more probable winning move. So, the program uses an algorithm for AI player to make the move that has a high probability

to win the game compared to other moves. For that, the program stores the move with high probability to win the game in a variable right after the moves are generated in succ(state) method. So, later that variable is used by the AI player to make a move. Out of all possible moves, the AI player prioritises the move that generates the score for it. That is the move including the heap with two sticks.

For instance:

```
state = ("max", 7, 2, 2, 0, 0)
```

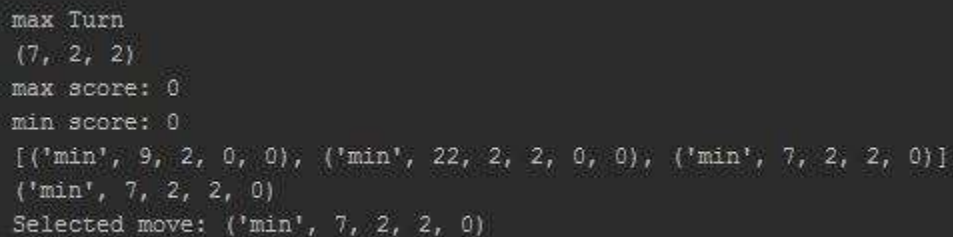
For above state, the list of all possible states is given below:

```
List = [('min', 9, 2, 0, 0), ('min', 22, 2, 2, 0, 0), ('min', 7, 2, 2, 0)]
```

In this case, AI player is 'max'.

Out of those three possible moves, the third move is selected by the AI player since it can gain 2 scores with that move.

The same condition was seen in the running program which is shown below:



```
max Turn
(7, 2, 2)
max score: 0
min score: 0
[('min', 9, 2, 0, 0), ('min', 22, 2, 2, 0, 0), ('min', 7, 2, 2, 0)]
('min', 7, 2, 2, 0)
Selected move: ('min', 7, 2, 2, 0)
```

Figure 1: Move made by AI player

If there is no score generating move in the list of possible states, then AI player selects the first element in the list which is made by placing a heap on another heap.

COMPARISON OF AI PLAYER WITH RANDOM PLAYER

The random player chooses the move randomly from the list of possible states whereas the AI player chooses the move considering that it has to generate a score for it. From various tests in the program, it was found that AI player performs better than a random player. The game option was selected to conduct a game between a random player and AI player (option 2 in the program)

Screenshots of one of the test are shown below:

```
Select Game Options:
1. Game between random and random player (Enter 1)
2. Game between random and AI player (Enter 2)
3. Game between human and human player (Enter 3)
> 2

mini random player
max: AI player

min Turn
(2, 2, 4, 2)
max score: 0
min score: 0
[('max', 6, 2, 2, 0, 0), ('max', 2, 4, 2, 0, 2), ('max', 2, 2, 2, 2, 0, 0), ('max', 2, 2, 4, 0, 2)]
Selected move: ('max', 2, 2, 2, 2, 0, 0)

max Turn
(2, 2, 2, 2, 2)
max score: 0
min score: 0
[('min', 2, 2, 2, 2, 2, 0)]
[('min', 2, 2, 2, 2, 2, 0)]
Selected move: ('min', 2, 2, 2, 2, 2, 0)

min Turn
(2, 2, 2, 2)
max score: 2
min score: 0
[('max', 2, 2, 2, 2, 2)]
Selected move: ('max', 2, 2, 2, 2, 2)
```

Figure 2: Screenshot 1

```
max Turn
(2, 2, 2)
max score: 2
min score: 2
[('min', 2, 2, 4, 2)]
[('min', 2, 2, 4, 2)]
Selected move: ('min', 2, 2, 4, 2)

min Turn
(2, 2)
max score: 4
min score: 2
[('max', 2, 4, 4)]
Selected move: ('max', 2, 4, 4)

max Turn
(2,)
max score: 4
min score: 4
[('min', 6, 4)]
[('min', 6, 4)]
Selected move: ('min', 6, 4)

Score of max: 6
Score of min: 4
Max won the game.
```

Figure 3: Screenshot 2

In above screenshots, it is proved that Max won the game which is AI player.

CONCLUSION

A game strategy for Super-Nim was successfully implemented in python. The moves selected by the random player and AI player were analysed, and it was found that AI player performs better than the random player in the game.