# Queen's Gambit

SOC-2024

Aditi Singh

June 2024

## Contents

# 1 Week1

## 1.1 Greedy or not

### 1.1.1 Overview

This is a simple game in which there is a list of n numbers and two players move alternately. On each move, a player removes either the first or last number from the list, and their score increases by that number and that number gets deleted. Both players try to maximize their scores and play optimally. A certain player wins if their score is strictly greater than the score of the other person.

### 1.1.2 My implementation

Initially for a long time I believed that optimal play for a person is that he decides between both the ends and pick the number which is greater. I wrote the code for this in `greedy_useless.py`. Eventually I realized that this is not optimal and then I implemented the recursive strategy. In this strategy there is a function named bestmove which returns a list which contains the bestmove and a projected score. This the score is further recursively calculated. My final code is in `greedy_working.py`. I eventually realized that the code is taking too much time for large arrays mainly because of it's exponential time complexity. I understood the process of implementing dynamic programming and I am yet to do that.

## 1.2 Nim

### 1.2.1 Overview

The number of piles, i.e., the rows in the image, of the game is finite, and each pile contains a finite number of matchsticks. These numbers can vary for different instances of Nim. Each player, during her turn, chooses exactly one pile, and removes any number of matchsticks from the pile she has selected (she must remove at least one matchstick). The player who removes the last matchstick wins the game.

### 1.2.2 Theory behind the Game

have my SOS on Game Theory due to which I have an idea on games like nim, Hackenbush and domineering. This past knowledge for nim carved my way easy for this.

So basically, for a particular Nim position we can calculate the bitwise XOR and if it is zero then the first player can force a win. If this is not the case then the second player can force a win.

The reason for the above logic lies in Bouton's Theorem which is basically that from a Nim zero position any move will lead to only a non-zero Nim position while there exists a move from a non-zero nim position to a zero position. This is simple to prove and easy to visualize.

### 1.2.3 My implementation

I first implemented a simple 2 player game in `nim_basic.py`. After this I implemented the game with computer with the above strategy in `nim_final.py` eventually realizing that I implemented the game in misere play(mainly because that the website link provided for playing also has misere play). So both the files above contain the gameplay in misere play wherein the person who picks the last looses the game.

Followed by this I created the gameplay in normal play that ther person who plays last wins. The implementation is done in `nim_normal.py`.

#### 1.2.4 QnA

**Question 1:** Does there always exist a sequence of moves such that the player making the first move wins?
**Answer:** No, the person making the first move wins only if the Nim position is not zero that is it's Bitwise XOR in non zero.

**Question 2:** Given a Nim position, is there a way to determine whether it is possible to win for any player with perfect play? If not, why? If yes, how?
**Answer:** Yes it is possible, If the position is a Nim zero position than player2 can guarantee a win otherise Player1 can guarantee a win.

**Question 3:** If a sequence of move which guarantees a win does exist for a Nim position,how will you determine the sequence of moves? That is to say, how is the computer coded to play the most optimum way?
**Answer:** That has been coded in nim_normal.py

### 1.3 Pawnscape

#### 1.3.1 Overview

In this game, you're provided with an NxN board where each player has N pawns positioned on both ends in the first and last rows. Victory is achieved by getting one of your pawns to the opposite end. If no feasible moves remain,the game results in a draw. Pawns are restricted to moving one step forward, even for their initial move.

#### 1.3.2 My implementation

I started playing with the helper code provided for n=3 and figured out that at any case the game ends on a draw. Subsequently I moved forward for n=4 but that seemed tedious however I managed to remove some repeated testcases and figure out the cases manually. I developed my own engine in `pawnscape.py` and tried with computing that with the help of engine but failed at the first instant. However I worked by printing stuffs and drawing conclusion manually.

#### 1.3.3 QnA

**Question1:** For n=4 find if there is a strategy that ensures that white never loses irrespective of what black does
**Answer:** Yes there always exist a strategy that white doesn't loose

**Question2:** For n=4 find if there is a strategy that ensures that white always wins irrespective of what black does. If the answer of the first part is NO then you can ignore this part
**Answer:** No, White can't always win. Many a times the game ends at a draw

**Question4:** Now play this game when you move second i.e. you have chosen the black pieces. For n=4 find if there is a strategy that ensures that black always wins irrespective of what white does. Ignore this if the answer of part 1 is YES
**Answer:** No, there isn't any such strategy

**Question5:** For n=4 find if there is a strategy that ensures that black never loses irrespective of what white does. Ignore this if the answer of part 2 is YES
**Answer:** Yes ,there are strategies that black ends up drawing the game or win up.

**Question8:** For those who are comfortable in Python/optional) Implement this game on your own in python. You can implement it in any way you like. Also code your strategies as well
**Answer:** Have implemented the working code in `pawnscape.py`. Will implement the strategies soon.

## 1.4 Chomp

### 1.4.1 Overview

Chomp is a two-player strategy game played on a rectangular grid composed of smaller square cells, resembling the blocks of a chocolate bar. Players take turns selecting and "eating" (removing) one block along with all blocks positioned below it and to its right. The top left block is "poisoned," and the player who consumes it loses the game

Figure 1: A basic gameplay for chomp

### 1.4.2 My implementation

I started with the observing some basic working and how the effective strategy works. However I came across the strategy-stealing argument while reading about Game theory eventually leading me to prove that Player 1 always wins. The argument is that if we assume Player2 has a winning strategy then if the 1st player just removes the lowermost block then player2 has a strategy to win. But this strategy can be implemented by Player 1 as well because whatever player2 played in the 2nd move was a possible move for player1 and the game could have just continued in the opposite way thus contradicting the original statement.

### 1.4.3 QnA

**Question1:** Does there always exist a sequence of moves such that the player making the first move wins?
**Answer:** Yes, there exists one which is stated above

**Question2:** Given a Chomp position, is there a way to determine whether it is possible to win for any player with perfect play? Can you deduce it?
**Answer:** Player1 always has a winning strategy nevertheless of what player2 does

**Question3:** If a sequence of move which guarantees a win does exist for a Chomp position, can you determine the sequence of moves?
**Answer:** I am on the way of coding and figuring this out. The above argument just proves player1 wins but doesn't provide a definite strategy