Epstein's Put or Take a Square Game

Ally Preston Student Number: 1501696

Supervisor: Ilia Krasikov

Department of Mathematics, College of Engineering,

Design and Physical Sciences,

Brunel University London

24^{th} March 2019

Contents

1 Introduction **1**

1.1 Rules of the game 1

1.2 Classification of numbers 2

1.3 Rationality and individual goals 3

2 Initial definitions and notation **3**

2.1 Types of numbers 3

2.2 More definitions 4

3 Programming approach **5**

3.1 Initial ideas 5

3.2 Algorithmic constraints 5

3.3 Direct 'tree' approach 6

3.4 Layer approach 10

3.5 Density of elements 15

4 Investigating patterns **20**

4.1 Primitive elements in layer 2 20

4.1.1 Generation of numbers from primitive elements 21

4.2 Sum and difference of squares 24

4.2.1 Sum of squares 24

4.2.2 Difference of squares 24

4.2.3 Difference between layer 2 elements 25

4.2.4 Conclusions for layer 2 26

4.3 Layer 3 patterns 26

4.3.1 Layer 3 differences 27

4.4 Drawing patterns 27

4.4.1 Special drawing case for 2 and 3 28

5 Other investigations **29**

5.1 Random moves 29

5.2 Constant moves 30

6 Conclusions **32**

A List of all numbers under 10,000 for the first 6 layers **35**

B Drawing positions for numbers from 1 to 100 that would result in a draw within the first 10 moves **36**

List of Figures

1 Tree showing possible moves for the number 4708 2
2 Output for the number 257, displaying all possible moves in all directions for 6 steps 9
3 Output for the number 4708, displaying all possible moves in all directions for 4 steps 10
4 Ordered W elements for the first 6 layers 14
5 Ordered L elements for the first 6 layers 15
6 The relative density of the W positions up to 1000 16
7 The relative density of the L positions up to 1000 17
8 The relative density of the D positions up to 1000 17
9 The relative density of the W positions up to 1000 in a logarithmic scale . 18
10 The relative density of the L positions up to 1000 in a logarithmic scale . . 19
11 The relative density of the W positions up to 1000 in a logarithmic scale . 19
12 differences between consecutive elements the first 100 elements from layer 2 25
13 differences between consecutive elements the first 100 elements from layer 3 27
14 Tree output for 8, showing 6 possible moves in all directions 28
15 Histogram showing the number of moves for random games to finish 30
16 Number of moves to obtain zero when only subtracting, for numbers 1 to 10000 31

Acknowledgements

I would like to thank Ilia Krasikov, without his help and guidance this project would have been impossible to attempt. I would also like to thank Natasha Trott for being very supportive throughout this project and this year in general.

Abstract

This project looks at exploring some investigative techniques for analysing Epstein's put or take a square game. Which involves starting with a positive integer and two players take it in turns adding or subtracting the largest perfect square smaller than the number with the goal of obtaining zero. Although no definitive answers have been found this project develops some useful programs for identifying numbers with specific properties toward the outcome of the game. Also considered are the patterns found for numbers that will result in a win or loss for a small number of moves and the presence of primitive elements that are able to generate further solution sets for the sets of numbers that will result in a win within a few moves. There is some considering the density of elements of each type and the rates at which they grow. Considered at the end are different methods of the game being played and identifying whether other interesting patterns can be found from this information.

1 Introduction

1.1 Rules of the game

Epstein's put or take a square game is an unsolved mathematical problem. The most comprehensive source is a section from the book 'Winning ways for your mathematical plays' P.484 [1]. There seems to not be any computational results on the web and it seems any patterns are too complicated at present to analyse at the moment. The basis of the game is two players start with a natural number N, each player then takes turns adding or subtracting the largest perfect square that is smaller than the current number. We can mathematically define the moves for a turn in the form,

$$N \to N - \lfloor \sqrt{N} \rfloor \text{ or } N + \lfloor \sqrt{N} \rfloor. \tag{1}$$

The first player to be able to reach the number 0 wins. This means that for a player to win they need to begin their turn with a square number.
Consider an example game beginning with the number 30.

1. The first player has to use the largest square below 30 which is 25, P_1 then decides to do $30 - 25 = 5$.

2. The second player has to add or subtract 4, however this has then forced P_2 into a position where either move will obtain a square number. (Adding yields 9 whereas subtracting yields 1)

3. Regardless of the second players action, the first player is able to subtract the remaining square number to obtain 0 and win the game.

This then forms the basis of the game. Although it is a rather simple premise it gets increasingly difficult when larger numbers are used or more moves are required to win. Let us consider for example 4708. This number happens to be one that will win in 4 moves for the second player from any of the following series of moves.

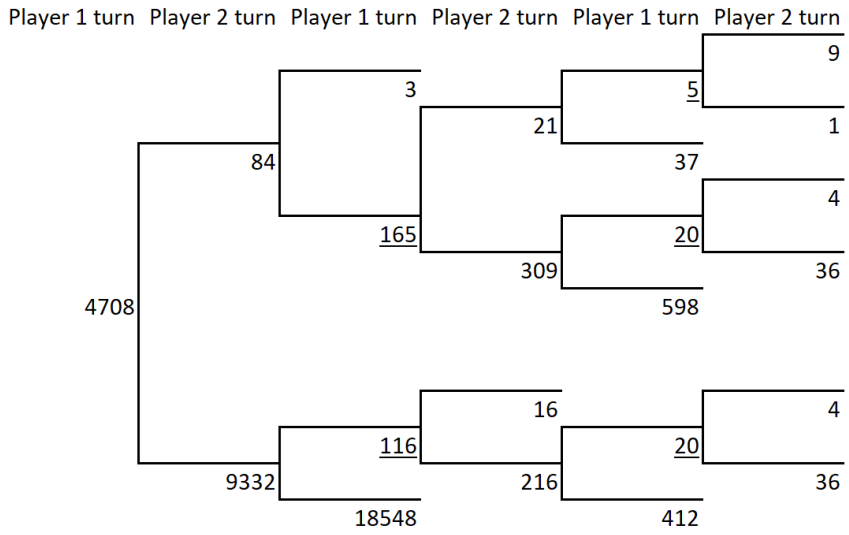

Figure 1: Tree showing possible moves for the number 4708

Figure 1 shows the possible moves for a game starting from 4708, where each column shows the number that each player will start with. The optimal moves for player 2, and the ones that are investigated, are shown with an underline. Here we can see no matter what move the first player makes if the second player plays optimally they will arrive at a winning condition of having a square number at the start of the turn. This shows that they cannot do anything other than lose in four moves (or less if they go for 16 instead of 216 on their second move).

There is also no obvious pattern for the results of each number, 4707 and 4709 have no special properties. This means both numerical and analytic investigation will be used at the start of this project.

1.2 Classification of numbers

The aim for this project will be to look at whether we can find an efficient algorithm or calculation to solve for each distinct number N, that will conclude either:

1. The number is a guaranteed win for the first player.
2. The number is a guaranteed loss for the first player.
3. The number will allow one of the players to force a draw by returning to a previously seen number.

4. The number will lead to a game which will continue infinitely without certainly reaching a win for either player or a draw.

1.3 Rationality and individual goals

For the purposes of this project we will initially begin by assuming people are completely rational in their approach. This means all players are able to calculate all possible outcomes and use this information to select the optimal move, resulting in the fastest win for them, if possible. Although unrealistic for an individual, it may be possible if we discover a method of grouping numbers.

Whilst an individual is playing this game they will prioritise obtaining the following outcomes; winning, drawing and then losing. This means that any computer simulation will first look for any possible wins. Then if no apparent win is possible they will instead select the first available draw possible. In the case that no win or draw is available they will select the moves that result in the longest time before a loss. In the case where we have an infinite series with neither player winning we will consider this to be a draw.

2 Initial definitions and notation

2.1 Types of numbers

The ultimate goal of this project will be to work out if it is possible to put numbers into a number of groups according to the outcome of the game. This can be done through the use of computer programming or manual calculations for ease of readability the groups will be focusing on the outcome from the perspective of player 1, therefore if player 2 would win we will consider this a loss for player 1. So far groups that we can observe are detailed in the table below.

W	Numbers that will result in a win for the first player if both players play optimally.
L	Numbers that will result in a loss for the first player if they both players optimally.
D	Numbers that will be able to repeat in a loop with other moves resulting in a loss for the player not causing the loop
D_∞	Numbers that are not members of any of the following groups and theoretically will have no optimal strategy for the number of iterations we are looking at, leading to an infinite series of moves with no forced draws, or wins.

Table 1: Types of numbers in our investigation

The first step in this project it to write some MATLAB programmes that can calculate numbers that are guaranteed winning and losing positions for a player through algorithmic processes. These codes and their resulting outputs are discussed in more detail in Section 5. The results from these programs can be used to build a set of numbers that belong to W and L. A similar algorithm can be used to construct a group for D by identifying repeated numbers. However, there is no theory which confirms that a number will not ever fall into one of these groups. Therefore we will work under the assumption that all numbers will tend to one of these groups eventually. However any numbers unresolved by the end of calculations will be designated into D_∞. Generally very little is known about this problem, and as calculations have only been done with small numbers [1]. We have no knowledge of whether $D_\infty = \emptyset$ or has any members at all.

2.2 More definitions

During our investigation it is useful to split numbers into sets of layers or l_i where i is the number of moves using optimal play to obtain a win. For example the number 30 would be a member of the l_3 group, as optimal play will result in a win in 3 moves. As we will be considering the moves from the starting player then this means on even turns it will be player 2's turn, and on odd turns it will be player 1's turn. This means the sets for W and L have sub members $W = \bigcup_{k=1}^{\infty} l_{2k-1}$ and $L = \bigcup_{k=1}^{\infty} l_{2k}$.

As the L layers will move to W and vice versa there is a loop between the layers. However the constraints are slightly different for the types of layers as players will be

picking the optimal move present. Due to this a number that has at least one move available that will go to a L position for it to be a W number. On the other hand a number will only be a L position if both moves available go to a winning position as if given the choice a player will avoid going to a W position. So the constraints for L numbers are far stricter than that of W numbers.

As the programs used will not continue for an infinite amount of time drawing possibilities, therefore when considering D positions it is only draws for the layers we have checked. Therefore some of the members of D_∞ for that layer will be members of D but have not been found. Therefore for D at l_k we will denote it D_k.

3 Programming approach

3.1 Initial ideas

For this project the first step was to utilise a computer to generate a list of numbers that fell into specific category. The idea of this program was that it could check a large set of data fairly quickly and then give us information that we could look at and apply different methods to investigate.

3.2 Algorithmic constraints

Due to limitations with computers we cannot look for an infinite range or series of moves. To prevent the number of moves becoming an issue we will be limiting each program to have an input of the number of moves we look in advance. An easy way to avoid the range becoming an issue is to only look for numbers that are in a range of 1 to αN, where α is an arbitrary fixed positive number and N is our original starting number, or upper limit of numbers we are testing if we are testing a range of numbers, this will hopefully help to eliminate any issue of having solutions for numbers outside the range we are investigating missed because we did not compute properly. For example say that we are testing the numbers from 1 to 100, we would want to not lose larger solutions because their solutions lie slightly outside the range, for example 95 may be a solution if it goes to 176. to avoid this issue we can introduce α as 100 and then look at the range of $\alpha N = 100 \times 100 = 10000$. When we then have our results for the program we diregard

anything outside our original range N, therefore hopefully preventing any of our numbers not being fully investigated.

3.3 Direct 'tree' approach

The first MATLAB program is a very simple style, it just started with a number and checked if it was a square. If it wasn't, added and subtracted the square below the number current number. This process was then repeated until the maximum number of iterations was complete or all numbers were either squares or numbers seen previously.

This then gives us a collection of all the numbers that could be generated from a single starting point. To represent the outputs in a tree structure another MATLAB code created by `Jean-Yves Tinevez` can be used [2]. The code for this can be seen below. We would have inputs of the number we start from N and number of moves *maxmoves*

```
if sqrt(N) == floor(sqrt(N)) %checks if N is an exact square
    P1W = [N 1]; %Player 1 wins immediately
else
    testnumbers = [N]; %adds the number to the testing box
end
```

This step just checks if the number we original started with was a square number, to make sure it doesn't run all the iterations when the solution was trivial.

```
while maxmoves > 0 %makes sure that the program doesn't get stuck
    for k = 1:length(testnumbers) % tests all the numbers in the test box
        A = testnumbers(k);
        %fetches the variable to work as the lower square
           if ismember(A,storage)
        else
        squarebelow = (floor(sqrt(A)))^2;
        %finds the square number that is below our test number
        A1 = A + squarebelow;
        A2 = A - squarebelow; %adds and subtracts the square below A
        [t] = t.addnode(nodecounter, A1); %adds the branches to our tree
        [t] = t.addnode(nodecounter, A2);
```

```
nodecounter = nodecounter + 1; %moves the plotting function along
```

This stage of the code adds and subtracts the squares above and below the number we are testing and adds them to our plotted tree.

```
if ismember(A1, storage)
  %checks if we have seen this number before and can cut the branch
     draw = draw + 1;
     storage = [storage, A1]; %adds our number into the storage
elseif sqrt(A1) == floor(sqrt(A1))
%checks if the number is a square
    square = square + 1;
    storage = [storage, A1]; %adds our number into the storage
else
    newnumbers = [newnumbers, A1];
    %adds our number into the new numbers box
end
if ismember(A2, storage)
%checks if we have seen this number before and can cut the branch
    draw = draw + 1;
    storage = [storage, A2]; %adds our number into the storage
elseif sqrt(A2) == floor(sqrt(A2))
%checks if the number is a square
    square = square + 1;
    storage = [storage, A2]; %adds our number into the storage
else
    newnumbers = [newnumbers, A2];
    %adds our number into the new numbers box
end
if square == 2 %checks if both numbers we found are squares
    if player == 1 %checks if it is player 1's turn
        P1W = [P1W; A turn]; %records the numbers
    else
        P2W = [P2W; A turn]; %records the numbers
```

```
            end
        end
        if draw == 2 %records drawn positions
            D = [D [A; turn]];
        elseif and(draw == 1, square == 1)
            D = [D [A; turn]];
        end
        stop = 0; % a variable to stop the loop going over
```

The code here checks the numbers we just found and allocates them to the correct groups

```
        while and(or(sqrt(t.get(nodecounter))==
        floor(sqrt(t.get(nodecounter))),ismember(t.get(nodecounter),storage))
        , stop == 0)
            if and(nnodes(t) == nodecounter,maxmoves ~= 1)
            %makes sure that the loop doesnt get too high
                stop = 1;
            else
                nodecounter = nodecounter + 1;
                %checks if the next number in the plot is a square or a draw
            end
        end
        end
    end
    storage = [storage, testnumbers]; %stores the numbers for this round
    testnumbers = newnumbers; %moves the new numbers to now be tested
    newnumbers = []; %resets the new numbers
    player = -player; %sets the player turn to the opposite player
    turn = turn + 1; %counts on the turn
    square = 0; %resets old variables
    draw = 0;
    maxmoves = maxmoves - 1; %records we made a move
end
```

The code then works out the exact places to plot the next branches on the tree and resets

variables for the next iteration.

The end result for this would then produce a tree of all the possible moves up to *maxmoves* To illustrate this we can use the example input of 257 to see and obtain the output shown in Figure [2].

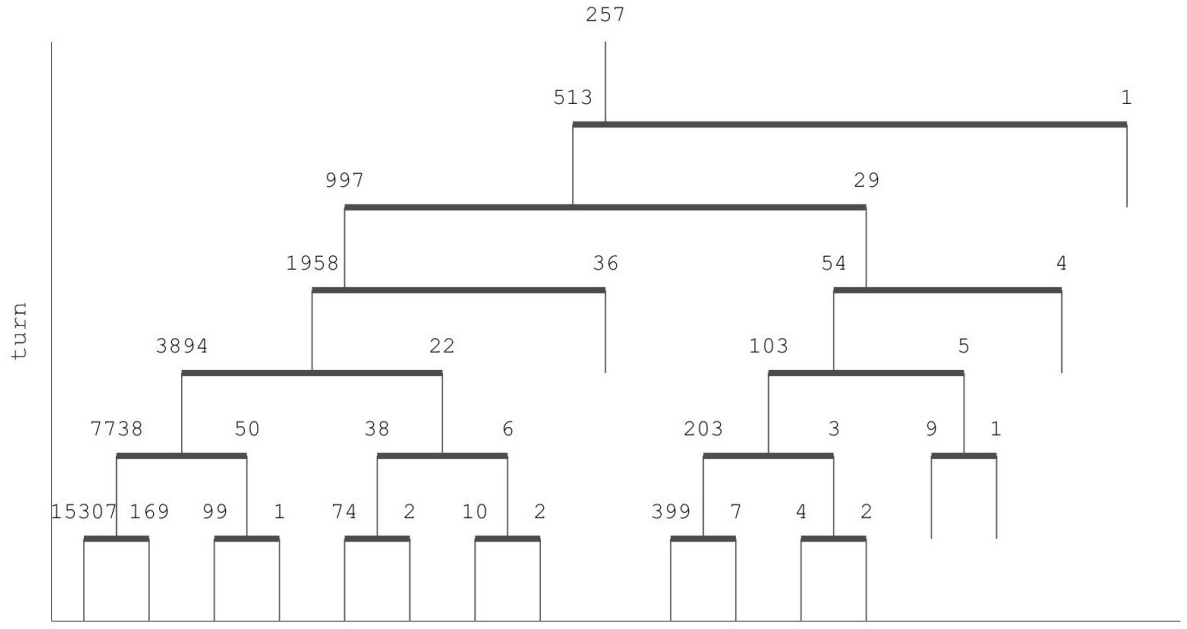

Figure 2: Output for the number 257, displaying all possible moves in all directions for 6 steps

From this figure we can see there is a potential winning path for player 2, as on each of player 1's moves they have the choice of a square number or a move that will lead to the win for player 2 As the first player is forced to make the move to 513 the second player will move to 29. The first player is then forced to make the move to 54, and the second player will then move to 5. This then leaves the first player in a losing position as either adding or subtracting will result in a square number for the second player to win with. We can also get an output tree for the number 4708 which we created manually earlier. Although due to limitations with the viewing of the output we can only run it for 4 moves as the display begins to overlap on the bottom left branches.

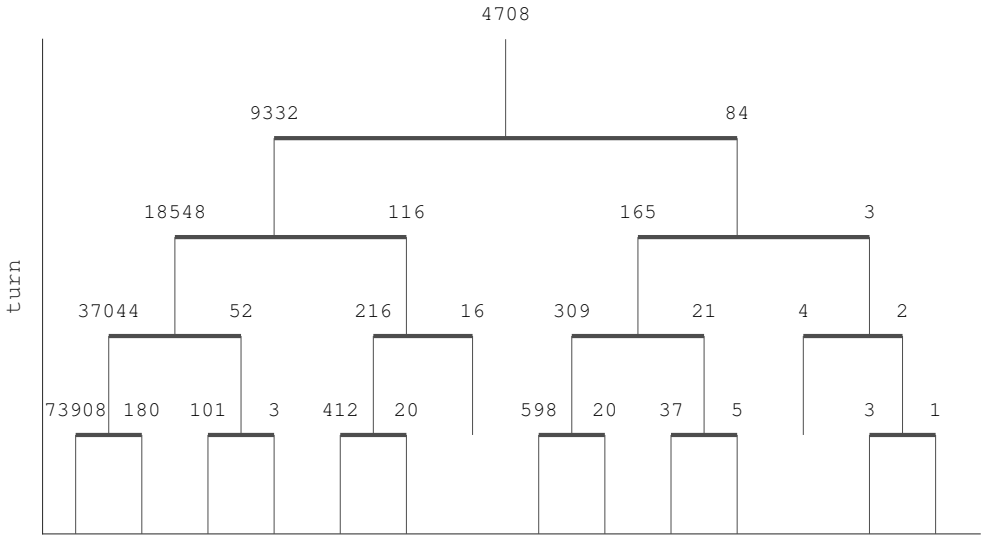

Figure 3: Output for the number 4708, displaying all possible moves in all directions for 4 steps

This program was a beneficial starting point for this project as we can use it to verify any positions we return to at a later state, as well as check the potential of any D numbers. However the program is rather ineffective for checking anything other than a single number as the output itself needs to be interpreted individually, resulting in using it being fairly slow.

An issue with this approach is that the number of calculations grows exponentially as theoretically each branch would have an additional 2 further numbers to calculate in the next round of moves. Due to some numbers being draws or square numbers the actual rate of new numbers needing to be calculated is slightly lower than 2, but through examples we can see that the rate of growth is exponential rather than linear. The issue is then that computers struggle to run this program for more that 6 iterations, as it would be checking up to $2^6 - 1 = 63$ different numbers. This then causes limits on the number of numbers we can check efficiently using the computers available.

3.4 Layer approach

The second series of programs created for this project was working backwards from known winning positions to find all the numbers that would result in a win if given an extra move. Given we can very easily see the numbers that will win in one move will simply be

square numbers. We can then look at other numbers and simulate a single move for both directions. If any numbers yield two moves from the winning layer then it will be a L position, as the only available moves for a player from this number will give a W position for the other player to use on their turn. We can then repeat this process of simulating moves in both directions for all numbers and see if any of these moves will yield a L position for either of the moves, we only need a single option for a losing move because if presented an option the optimal move will always be the one that gives the other player a L number. Hence this process can be repeated as many iterations as required to build up a set of layers that will result in wins, or losses, in as many moves as we iterate. We can also use this program to start identifying some of the drawing positions by extending the code for losing positions to look at the drawing positions for numbers. A number would be a draw if it is returning to a previous number found that the only other move available would result in a loss for that player, as the optimal move will be to return to the previous position continuously, hence causing a draw to occur. This can be done at during the iterations for the losing positions, if there is only the option of a draw, or a losing position for a number it will be saved as a draw.

For this code we will be checking numbers for the range of 1 to an upper limit specified N, we will then allow the program to run for all numbers up to a set limit higher than this number which we can specify as α. The program will check all numbers up to the limit of αN but will then not output the ones outside our initial range, as we will have not checked most of the numbers higher than those so they are not going to be accurately checked.

The program created will initially make a list of all the numbers that are squares in our range, this is denoted layer 1 and is saved into the winning positions. This can be done using this segment of code:

```
for loop = 1:length(testnumbers)
x = testnumbers(loop); %saves current number to make calculations easier
if floor(sqrt(x)) == sqrt(x) %checks if number is square
    P1W = [P1W, x]; %stores all the winning positions for player 1
end
```

We then can make a loop to run for the maximum number of iterations and find the winning and losing positions.

```
while iteration < maxiterations
    P2W = [P2W, turn];%records which turn we saw the numbers
    for loop = 1:length(testnumbers)
        A = testnumbers(loop); %pulls the number we are checking
        squarebelow = (floor(sqrt(A)))^2;
        %finds the square number that is below our test number
        A1 = A + squarebelow;
        A2 = A - squarebelow; %adds and subtracts the square below A
        t1 = ismember(A1, P1W);
        t2 = ismember(A2, P1W);
        %tests variables here to improve efficiency
        if and(t1, t2) %checks if both of the numbers go to a win
            P2W = [P2W, A]; %saves the number as a winning position
        elseif and(t1, ismember(A2, startingnumbers))
            Draw = [Draw, A]; %saves the number as a drawing position
        elseif and(t2, ismember(A1, startingnumbers))
            Draw = [Draw, A]; %saves the number as a drawing position
        end
    end
    iteration = iteration + 1; %counts on an iteration
    count = 1; %resets the variable
    while count < length(testnumbers)
    %deletes all the entries we have found that are into a group
        if ismember(testnumbers(count), P2W)
            testnumbers(count) = []; %deletes the entry if needed
        end
        count = count + 1; %counts on
    end
    turn = turn - 1; %records a turn has passed
```

First we look at the second player's turns in the iteration, as the preliminary stage does the first player's turn, if they only have moves that take to square numbers it will be a loss for them and it records the position that would yield this result. If they are offered

a losing position and a draw then they will opt for the draw, this is then recorded.

```
    P1W = [P1W, turn];%records which turn we saw the numbers
    for loop = 1:length(testnumbers)
        A = testnumbers(loop); %pulls the number we are checking
        squarebelow = (floor(sqrt(A)))^2;
        %finds the square number that is below our test number
        A1 = A + squarebelow;
        A2 = A - squarebelow; %adds and subtracts the square below A
        if or(ismember(A1, P2W), ismember(A2, P2W))
        %checks if one of the numbers go to a win
            P1W = [P1W, A]; %saves the number as a winning position
        end
    end
    count = 1; %resets the variable
    while count < length(testnumbers)
    %deletes all the entries we have found that are into a group
        if ismember(testnumbers(count), P1W)
            testnumbers(count) = []; %deletes the entry if needed
        end
        count = count + 1; %counts on
    end
    turn = turn - 1; %records a turn has passed
    iteration = iteration + 1; %counts on an iteration
end
```

The iteration then looks at the first players turns, if P_1 has an option of a winning move then they will select that move, all other moves will be ignored, as they will not force a draw as there is not a guaranteed losing position for the first player in this case.

The code then slices the resulting matrices to remove all numbers that are bigger than our initial starting range 1 to N and receive our 'lists' of numbers for winning and losing numbers, with our entries separated with a negative number denoting the layer.

This program was very efficient at testing a large batch of numbers very quickly and building a set of layers fairly quickly. We can use it to obtain results that were also found

by Richard guy in his book [1]. We can then use these numbers found to try to identify patterns found and potentially begin solving the problem. If we run it for $N = 10000$, $\alpha = 100$ and `maxmoves` $= 6$ we get a fairly large set of numbers which we can then use during later investigations.

The results for the first few layers are the following:

- Layer 1 - All Square numbers
- Layer 2 - 5, 20, 45, 80, 145, 580, 949, 1305, 1649, 2320 ...
- Layer 3 - 11,14,21,30,41,44,54,69,86,105,120,126,141,149 ...
- Layer 4 - 29, 101, 116, 135, 165, 236, 404, 445, 540, 565 ...

A much more complete list can be found in the appendices of numbers found in the investigation. If the elements are are plotted in the order that they appear we obtain the following graphs for the odd layers, and the even layers.

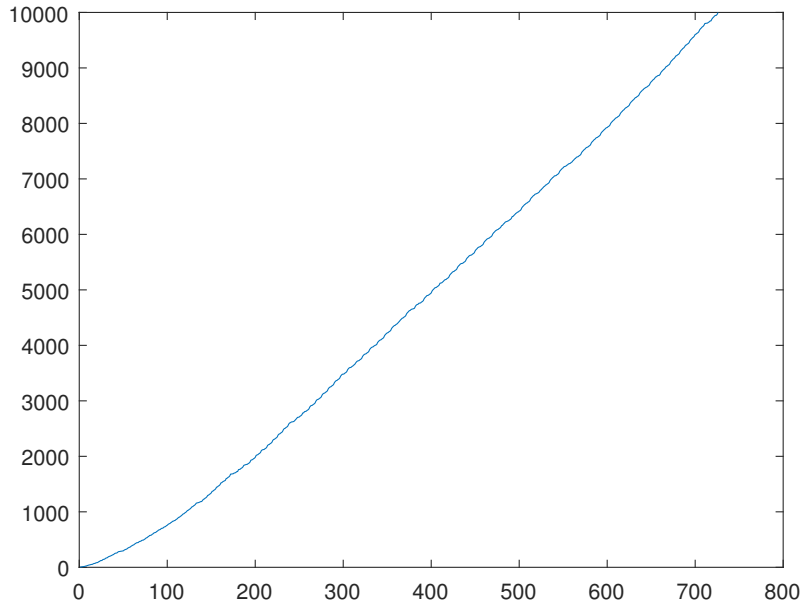

Figure 4: Ordered W elements for the first 6 layers

This graph shows the values of the first 300 winning positions for the first 6 layers, with the ordering of the elements on the x axis and their value on the y axis. As we can see the values for these grows exponentially, with the 100^{th} element being approximately 1000.

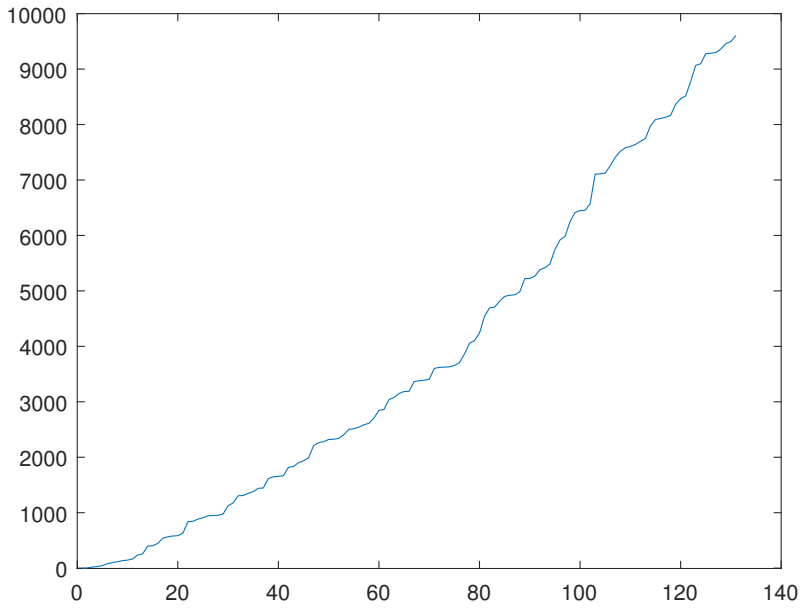

Figure 5: Ordered L elements for the first 6 layers

This graph shows the first 30 elements for L positions, as we can see the growth of the size of the elements is far faster than the W elements, reaching approximately 100000 by the 20^{th} element. The trend seems less smooth, but that may be due to the lower number of elements available, as the higher number of W elements gives more points to be plotted, hence giving a smoother curve.

We can see from the number of elements in even layers are significantly lower than the number of elements in the layers with odd numbers. This is because the requisite for odd layers, 'losing positions' have a requirement of both moves going to the layer below, whereas even layers only need one move to go to the layer below. This probably implies that in a game with two players, the player going first would have a significantly higher probability of winning the game from a random starting number, although we do not have any rigorous proof for this.

3.5 Density of elements

As we can see from our programs the density of even layers is lower than that of odd layers. we can then look at the number of elements in each of these categories using a

MATLAB script which computes,

$$\frac{\#W \leq N}{N}, \tag{2}$$

and,

$$\frac{\#L \leq N}{N}, \tag{3}$$

For a given N we then plot the results. This then yields the following graphs for the densities of W, L and D positions for the first two layers of the algorithm.

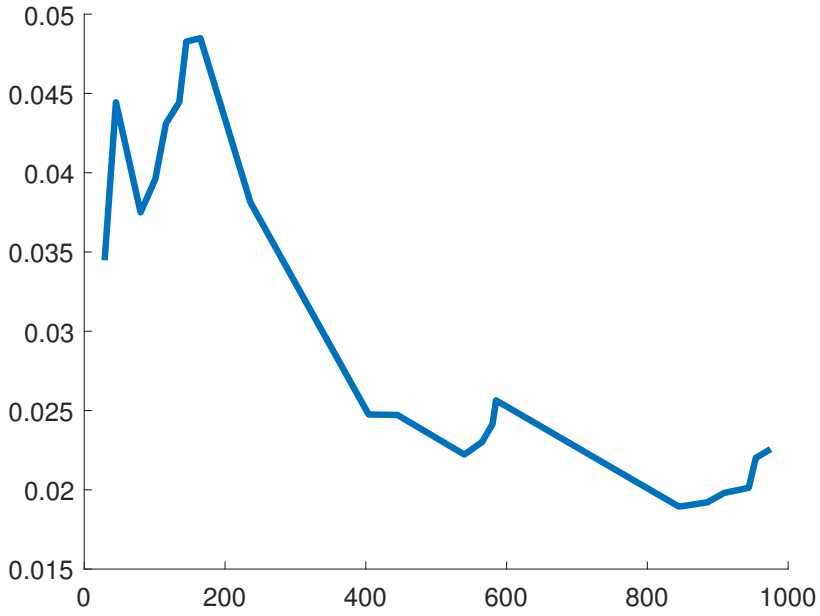

Figure 6: The relative density of the W positions up to 1000

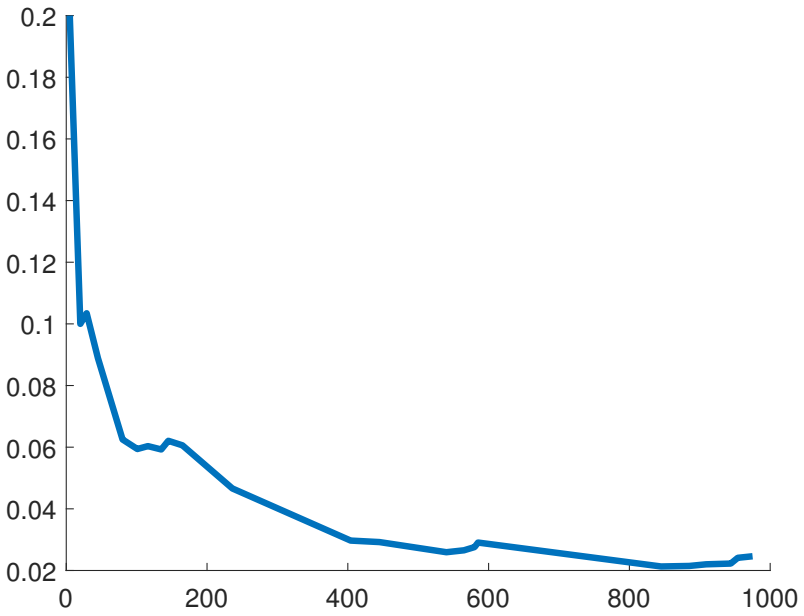

Figure 7: The relative density of the L positions up to 1000

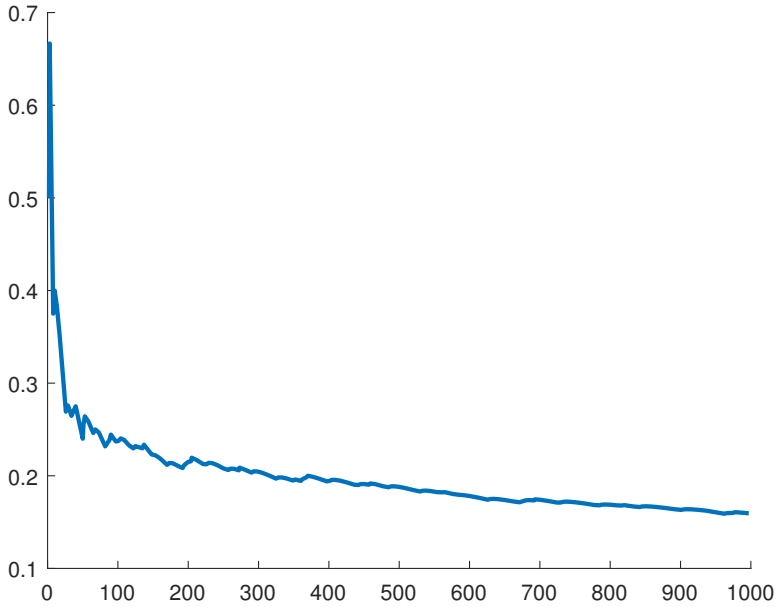

Figure 8: The relative density of the D positions up to 1000

For the first graph it makes sense to remove the first few elements as the density is abnormally high, as 1 is an element of that group so therefore the graph would have a starting density of 1.

We can see the overall densities diminish as we increase N for both cases, albeit slower for winning positions. This can be attributed to the rate at which square numbers grow. Our range interval for N is in the form $k^2 < N < (k+1)^2$ which gives us a range for the value of N of width $(k+1)^2 - 1 - k^2 = 2k - 1$, as $k \approx \sqrt{N}$ this means our interval will increase as N gets bigger. However the number of square numbers grows slower, this then means the higher you go the less dense the number of square numbers are.

If we then transform each of our graphs to obtain ones that are closer to linear trends, this then allows us to approximate the rate of growth for each of the sets. If we use the transformation for the density $density \to (\log density)^{-}1$ for the first two graphs and $density \to (\log density)$ for the third we obtain the following graphs.

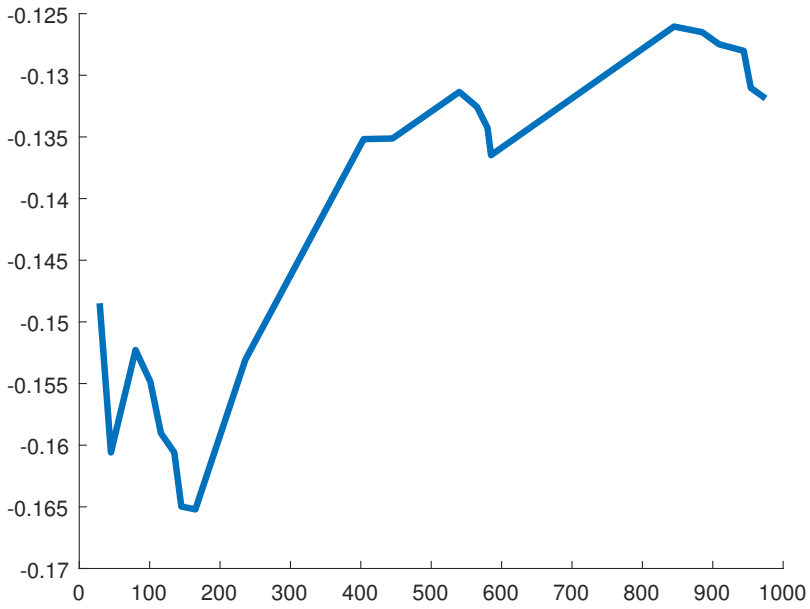

Figure 9: The relative density of the W positions up to 1000 in a logarithmic scale

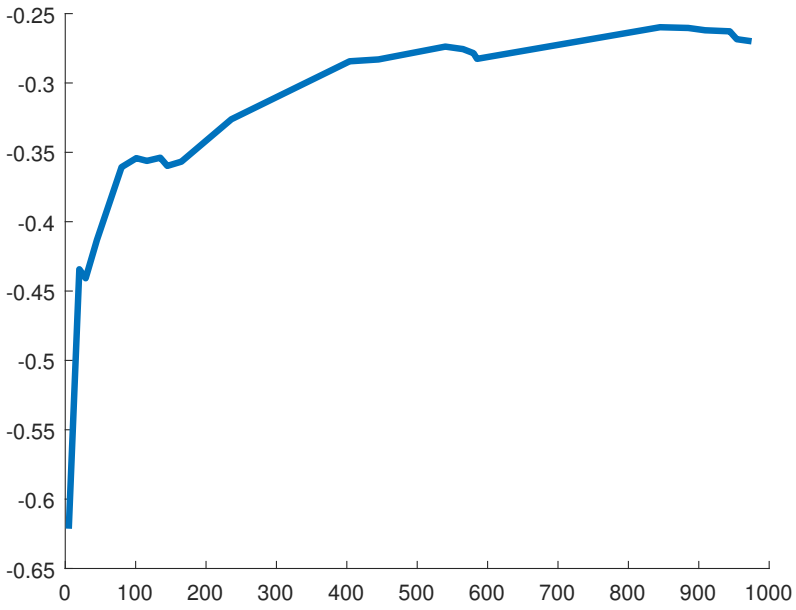

Figure 10: The relative density of the L positions up to 1000 in a logarithmic scale

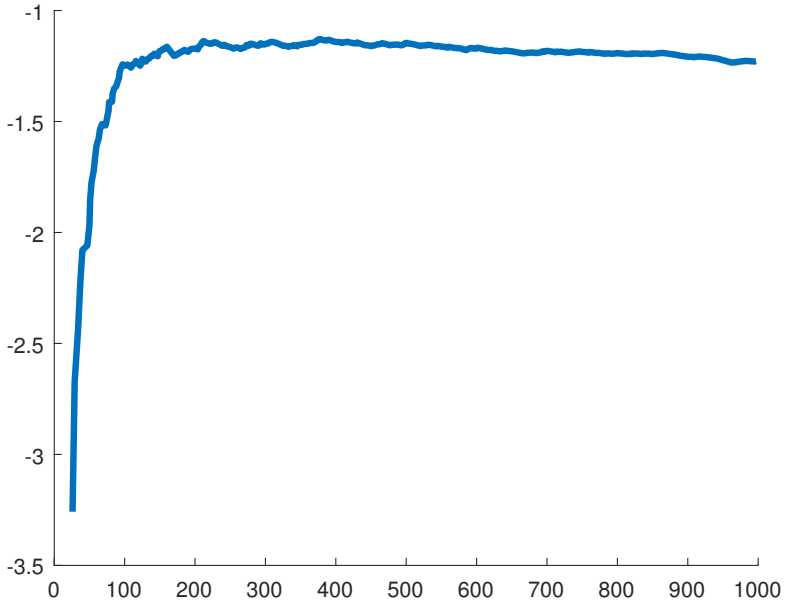

Figure 11: The relative density of the W positions up to 1000 in a logarithmic scale

As we can see, the graphs for both W and L grow slower than exponents, meaning that density is decreasing faster than the square root gap, however the densities of drawing patterns, for larger numbers is a constant, meaining that the rate at which drawing numbers are found is roughly constant.

4 Investigating patterns

As we now have a set of members from the second and third layer, as well as the densities for each we can look at them a little more analytically and begin to see if there are any patterns present.

4.1 Primitive elements in layer 2

From the results obtained from layer 2 we can see a few cases of notable interest. The first few members of the set are 5, 20, 45 and 80. This is interesting because they can be expressed as $5, 2^2 \times 5, 3^2 \times 5$ and $4^2 \times 5$. This means from a member of the layer 1 there are several other solutions generated from this element. This pattern does not happen for all elements but does for some, for example 979 does not generate any other members but a later element 4801 does. If we then only look at the elements that cannot be expressed in the form $W_i K^2$ where W_i is a previously seen element and K is an integer we limit the number of elements in the set, we obtain the following set of numbers, which would generate the other numbers shown.

Primitive element	numbers generated by primitive element
5	20, 45, 80
145	580, 1305, 2320, 3625, 5220, 7105, 9280, 14500, 17545, 20880, 24505, 28420, 32625, 37120,41905, 46980
949	$\emptyset$
1649	$\emptyset$
4901	19604, 44109
31025	$\emptyset$

Table 2: Primitive elements and numbers generated from them up to 50,000

The table shows the primitive elements and numbers generated by them. As we can see not all primitive elements are generators of other elements in this range, and the number of elements generated is not consistent.
This shows that these primitive elements are far more sparse and more evenly distributed than other elements in this layer.

If we then look at just their largest perfect square smaller than each element, the

number we would obtain from doing $N - X^2$ where $X = \lfloor \sqrt{N} \rfloor$, we obtain quite an interesting set of results shown in the table below.

Primitive element (P)	Square below (S)	(P) - (S)
5	4	1
145	144	1
949	900	49
1649	1600	49
4901	4900	1
31025	30976	49

Table 3: The difference between primitive elements and the square below them

This implies that all the primitive elements are not much larger than a square number, however if these elements were randomly distributed then they would be in the range $k^2 < N < (k+1)^2$, giving a width of the range to be $(k+1)^2 - k^2 = 2k + 1$, hence we would assume that on average the lower square would be $\frac{2k+1}{2}$ away from our chosen element N. So on average the distance would be roughly $\sqrt{N}$, but it is much smaller in practice, much closer to $N^{\frac{1}{4}}$. This will be analysed in the following section.

4.1.1 Generation of numbers from primitive elements

The fact that these primitive elements result in other solutions that are direct multiples can be attributed to the Pell's Equation [3]. Which is an equation in the following form,

$$x^2 - ny^2 = 1, n \in \mathbb{Z}. \tag{4}$$

The particular equation is useful as we have the variables in the form for our problem if we have a primitive element X,

$$X - k^2 = 1, \tag{5}$$

$$X + k^2 = m^2, \tag{6}$$

Where $k = \lfloor \sqrt{X} \rfloor$. If we eliminate X from the equations by subtracting (5) − (4) we get the equation.

$$m^2 - 2k^2 = 1. \tag{7}$$

Which is in the form of Pell's equation for $n = 2$. This is then a standard result we can use the solutions for to obtain $m = 3, k = 2$ [4], which are the two smallest solutions to the equation. These then go back into our original equation to obtain $X = 5$ Which is our first primitive element. The full generation of solutions for this are generated using the following equations, [4]

$$m_n = \frac{1}{2}\big((3 - 2\sqrt{2})^n + (3 + \sqrt{2})^n\big), \tag{8}$$

$$k_n = \frac{(3 + 2\sqrt{2})^n - (3 - 2\sqrt{2})^n}{2\sqrt{2}} \tag{9}$$

$$n \in \mathbb{Z}. \tag{10}$$

This then gives us some of the potential elements for our first layer that have a lower square of 1. The elements from Table 3 that have this property at the numbers 5, 145 and 4901; these happen to also be the numbers generated by the values of $n = 1, 2, 3$ for equations (5) and (6).

We can also find solutions for the other elements, which have a lower square of 49 rather than 1, a Pell's solution for this case can be obtained and used to generate these elements in a similar method.

The number of elements that will be generated from a primitive element can be described using the distance from the biggest square number below the element. If we express our primitive element X in the following form,

$$X = s^2 + r^2, \tag{11}$$

For X to be a primitive element the remainder portion must be a square. As r is the part of the number that is above the lover square it must be smaller than the width of square numbers, therefore it must be in the form $1 < r^2 \leq 2k$ for k^2 to still be the largest square below X. We then have the elements being generated from this primitive element in the following form.

$$i^2X - i^2k^2 = i^2r^2, \tag{12}$$

$$i^2X + i^2k^2 = i^2m^2, \tag{13}$$

for $i \in \mathbb{N}$. Using the separation of the square and remainder for X we obtain the following equation,

$$i^2X = i^2(s^2 + r^2). \tag{14}$$

Then putting this into equation (12) we then get an equation in the following form,

$$i^2(s^2 + r^2) - \lfloor\sqrt{i^2(s^2 + r^2)}\rfloor^2 = i^2r^2. \tag{15}$$

Which can simplify to,

$$i^2s^2 = \lfloor\sqrt{i^2(s^2 + r^2)}\rfloor^2. \tag{16}$$

If we then square root both sides and remove the floor function by changing to an inequality the following equation is found,

$$is + 1 > i\sqrt{s^2 + r^2}. \tag{17}$$

We then square both sides and obtain a quadratic inequality for i,

$$i^2s^2 + i^2r^2 < i^2s^2 + 2is + 1 \tag{18}$$

$$i^2r^2 - 2is - 1 < 0. \tag{19}$$

Which then can be solved using the quadratic formula to give an exact inequality for i,

$$i < \frac{s + \sqrt{s^2 + r^2}}{r^2} \tag{20}$$

This then means that for a number with a small difference to square number below it there is a long sequence of generated elements as the length of the series will be roughly $\frac{2s}{r^2}$ when r is small, this is seen as 5 and 145 generate long series of elements, but 949, with a larger r does not generate any.

The issue with using this method is that the lower square is not consistent and shows no pattern, so although we can find some of the elements using this method, we need to know which Pell's equation solutions to use in order to obtain solutions for our primitive elements in the first layer.

4.2 Sum and difference of squares

As we are looking for solutions in the form $X - k^2 = p^2$ and $X + k^2 = q^2$ where $k = \lfloor \sqrt{X} \rfloor$ we can do a little rearranging to obtain equations that are in a slightly better format.

$$X = p^2 + k^2 \tag{21}$$

$$X = q^2 - k^2. \tag{22}$$

Both of these equations are the sum and difference of squares respectively, and we can investigate some of the theory surrounding them and if any of them yield important results.

4.2.1 Sum of squares

Looking only at equation (21) for expressing X as the sum of two square numbers we can then build a set using the properties of the sum of squares. There is a theorem by Fermat, stating that a number can be expressed as a sum of two square numbers if and only if its prime factors in the form $4n - 1$ are to even powers [5]. This is interesting as all of our primitive elements are in this form, and thus can be expressed using the values of p and k found as solutions to the Pell's equations. However we run into the issue that although all the solutions have this property, not all numbers with this property are solutions. We can then create a simple program can be created to generate all the numbers that can be expressed as the sum of two squares by adding together two squares and recording the results for all square numbers in a range of 1 to N. The output after eliminating duplicates is shown below.

```
2,5,8,10,13,17,18,20,25,26,29,32,34,37,40,41,45,50,52,53,58,61,65 ...
```

These contain the results for out first layer elements, however some additional ones, as this is only numbers that have one property for our set of numbers from layer 1.

4.2.2 Difference of squares

If we now use equation (22) we can now try expressing the numbers as a difference of two squares,

$$X = q^2 - k^2 = (q - k)(q + k). \tag{23}$$

If we then consider the possibilities of whether q and k are even and odd numbers.

q	k	X
even	even	$(2a-2b)(2a+2b) = 4(a^2-b^2)$
even	odd	$(2a-2b-1)(2a+2b+1) = 4(a^2-b^2-b)-1$
odd	even	$(2a+1-2b)(2a+1+2b) = 4(a^2-b^2+a)+1$
odd	odd	$(2a-2b)(2a+2b+2) = 4(a^2-b^2+a-b)$

Table 4: Permutations for difference of two squares

This is interesting because all the options are either multiples of 4 if even which we can use to eliminate some of the numbers found in the set before, but it does not fully limit the group to only numbers which are in our l_2 set.

4.2.3 Difference between layer 2 elements

Another interesting result found whilst investigating the results from layer 2 is the difference between the elements in this layer. If we plot the difference between all the consecutive elements in this layer we find the following graph.

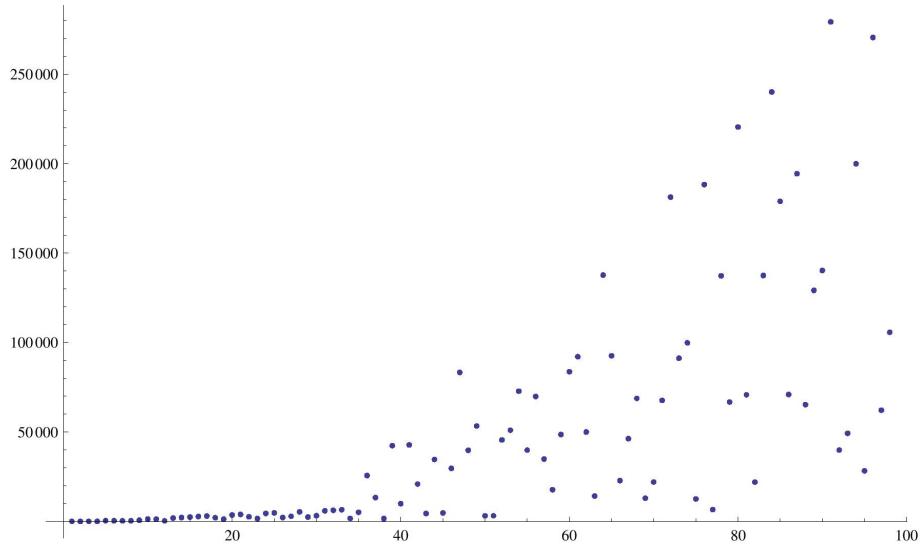

Figure 12: differences between consecutive elements the first 100 elements from layer 2

Which is interesting as all the differences a very small relative to the size of the elements, the 100^{th} element is just under 5,000,000, so almost 20 times the distance between the previous element. We can also see the rate at which these differences grow is pretty consistent as well, with the elements being more spaced out as they get bigger,

which is to be expected as the gaps between square numbers also increases at a similar rate.

4.2.4 Conclusions for layer 2

Unfortunately, although there are some ways to limit this group, there is not enough to obtain the exact properties for numbers that are exclusively in this group.

4.3 Layer 3 patterns

As there are no definitive patterns found for layer 2 it is a little hard to construct any meaningful interpretations for any subsequent layers. Looking at the whole layer we have the following numbers:

```
11,14,21,30,41,44,54,69,86,105, 120, 126, 141, 149, 164, 174, 189, 201,
216, 230, 245, 261, 276, 291, 294, 309, 329, 344, 366, 381, 405, ...
```

Which do not seem to have any obvious patterns at first, however upon further investigation there are some interesting properties, including the presence of primitive elements and some some interesting patterns in the difference between the elements.

Although we can't identify all the primitive elements for this layer as easily as layer 2, we can see some trends that imply the existence of such elements.
If we divide all the elements by the first element 11, we get very few elements that are still whole numbers, if this layer was truly random we could assume that $\approx \frac{1}{11}$ of the total elements would be multiples of 11, however this is not the case as there seems to only be 3.23% of elements are multiples of 11 for the elements under 1000. If we repeat this process we obtain the following percentages for each element.

Element	11	14	21	30	41	44	54	69	86	105
Percentage	3.23	12.90	17.74	6.45	6.45	1.61	3.23	6.45	4.84	6.45

Table 5: Percentage of elements in layer 3 that are multiples of each element in layer 3

This is interesting as it shows that elements are not uniformly distributed across all multiples, although some of these numbers are likely to be due to random chance as seen in layer 2, the majority of the elements do seem to show properties of generating other

elements, but we cannot really investigate as in depth as with layer 2 due to lack of theory or any meaningful equations that can be derived from these numbers.

4.3.1 Layer 3 differences

If we repeat the process as done for layer 2 of finding and plotting the difference between consecutive elements we obtain the following graph.

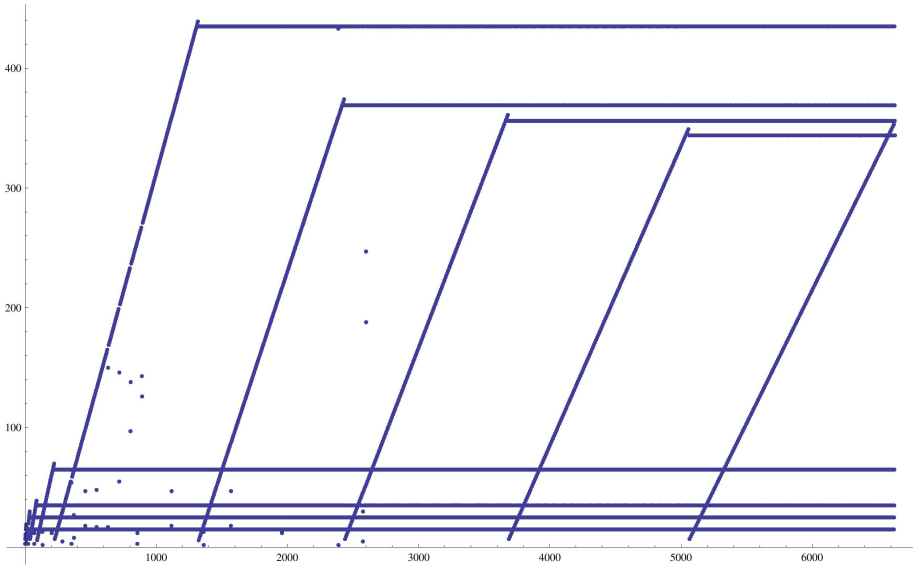

Figure 13: differences between consecutive elements the first 100 elements from layer 3

This is very interesting as most of the elements have a distance to the next element that follows a one of a series of patterns. Which is interesting because it suggests the pattern for these elements is pretty consistent. However at this time we cannot really interpret these patterns meaningfully for the generation of the elements as the order for each of the patterns seems fairly random when looking at a few key cases.

4.4 Drawing patterns

The number of D is much higher than the other types of elements (8), this makes it a little harder to identify patterns than done in sections 4.2 and 4.3. The interesting thing about this set is that there does not seem to be any form of primitive elements, with all numbers appearing to be multiples of each other in a seemingly random style. If we run the 'layer' program for $N = 1$ to 100 with $\alpha = 100$ for 6 iterations we obtain the following drawing positions:

```
2, 3, 7, 8, 10, 12, 13, 17, 26, 29, 34, 37, 40, 50, ...
```

A full list can be seen in the appendix. From just looking at the numbers it is a little difficult to interpret why they are a drawing position, if we consider an example with 8 using the 'Tree' program used earlier we obtain the following output which illustrates the series of moves that can be taken.

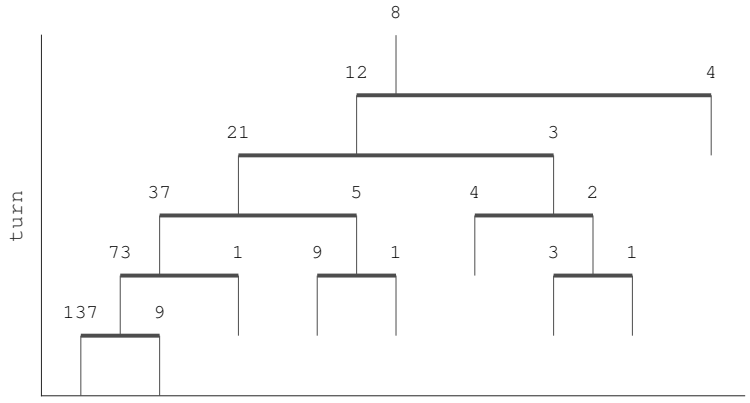

Figure 14: Tree output for 8, showing 6 possible moves in all directions

So for the number 8 we can see that the series of moves has to be $8 \to 12 \to 3$ as any other series of moves will lead to a loss for P_1 at 4, and subsequently a loss for P_2 at 5, this then leads us to 3, which is a special case which will be talked about in the following subsection. When we run the Tree program for other D elements we obtain similar patterns of the numbers having a short series of 'forced' moves to avoid losses followed by leading to the number 2 or the number 3. As this is very difficult to form equations from there are no further investigations that can be attempted at this time.

4.4.1 Special drawing case for 2 and 3

Usually we can consider a drawing sequence in a game to be 'undoing' the opponents last move, for example if P_1 added it would be countered by P_2 subtracting. However because of the nature of *Epstein's game* the option for the moves are likely to be different for each player. If P_1 does $N_1 + \lfloor\sqrt{N_1}\rfloor = N_2$ then the new largest perfect square below N_2 will be different unless a special case is met,

$$N - k^2 > (k-1)^2, \tag{24}$$

$$N + k^2 < (k+1)^2, \tag{25}$$

Where both N and k are positive integers. If we simplify then we get the following equations,

$$N > 2k^2 - 2k + 1, \tag{26}$$

$$N < 2k + 1. \tag{27}$$

The only times these conditions are met is when $N = 2$ or $N = 3$. As we get larger values of k subtracting a $\lfloor\sqrt{N}\rfloor$ will always change the largest perfect square smaller than N therefore the only solutions are our special cases for drawing 2 and 3.

5 Other investigations

As Direct investigations into the layer 2, layer 3 and drawing layers have not proven very successful and not obtained any results to work with the next stage in the investigation is to look at specific cases and see what happens and look at the results for any other data that can be found.

5.1 Random moves

If instead of the players looking for an optimal strategy the players instead play completely randomly from a starting number we can then see a few results. If we look at play for the numbers from 1 to 1000 and try each number 100 times, looking at most 25 moves in advance we obtain a ratio of the number of wins for player 1: the number of wins for player 2 equal to 0.9944 meaning player 2 actually seemed to win more of the games. Which is interesting because it seems that player 1/2 has a higher probability of winning the games, despite the number of l_{2k} elements being significantly lower than l_{2k+1} the ratio of wins is not exactly even but fairly close.

We also can see the turn that players won on in the following histogram.

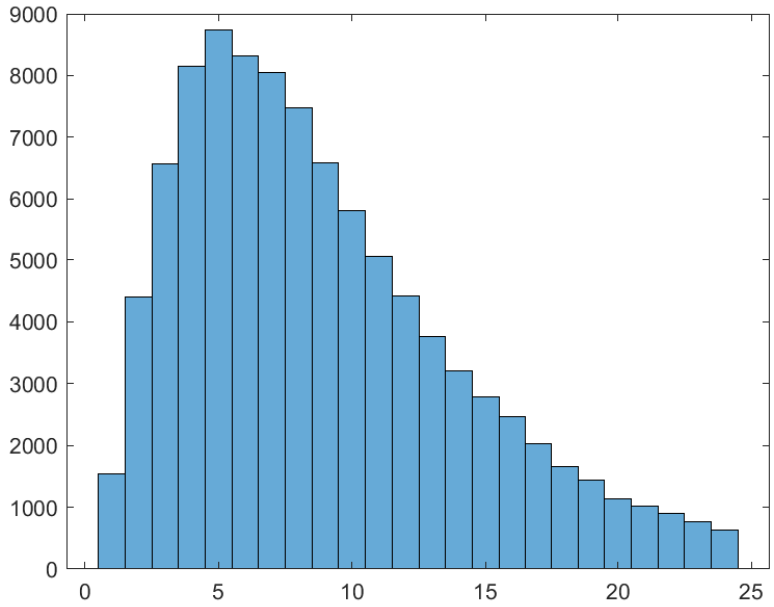

Figure 15: Histogram showing the number of moves for random games to finish

From the histogram we can see that for low number of games finished in a small number of moves, this is to be expected as it is unlikely that the random play will yield the optimal strategy for the members of those layers. It does seem though that the majority of games finish within 10 moves, meaning numbers are not too complicated to deal with and games do conclude within a reasonable amount of time.

Of the games played in this simulation only 3108 games exceeded the 25 move limit, which is a small fraction of the 100,000 games that were simulated in the investigation. This behaviour is expected though as some games will continue to loop round and do not make progress very quickly, hence timing out with the 25 limit. If we were to extend the limit to 100 moves we find none of the rounds are unresolved and the pattern for the number of moves is similar to the one shown in (15), with the ratio of player 1 wins: player 2 wins being slightly closer to 1 at 0.9978, still favouring player 2 though.

5.2 Constant moves

If instead of the players having a choice of adding or subtracting the moves instead followed a fixed pattern. If we consider the game being played where both players just subtract the largest perfect square smaller than the number we can record the number of moves it takes to reach zero. Using a code to simulate these games from 1 to 10000 we obtain the

following number of moves, arranged into a histogram for readability.

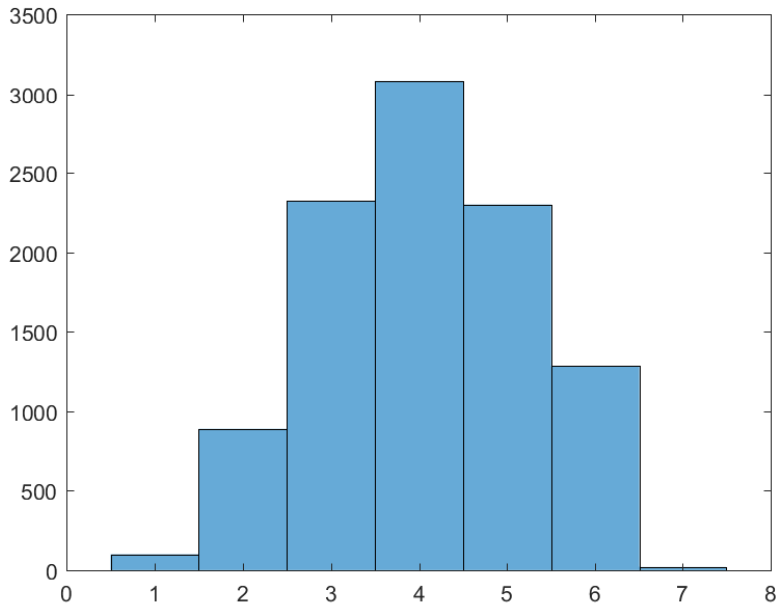

Figure 16: Number of moves to obtain zero when only subtracting, for numbers 1 to 10000

As we can see the number of moves forms a bell shaped distribution centred around 4. What is more interesting is the fact that no numbers in the range of 1 to 10000 took more than 7 moves to resolve. Suggesting that the number of moves is never going to exceed a limit for a given number. Upon further examples, this number appears to be $3 + \log N$ of the upper limit, as numbers up to 100 take up to 5 moves, 1000 takes up to 6, and 10000 takes 7 moves.

Another case to consider is if the players make opposite moves each turn, to simulate this we can use a code that alternates the players adding and subtracting the square. To make sure the game is winnable for the adding player they will subtract a square if and only if it is a square number at the start of their turn. For the first iteration we have P_1 adding and P_2 subtracting we yield 4170 wins for player 1, 4313 wins for player 2, and the remaining 1517 were draws. If we invert the game and have P_2 adding and P_1 subtracting we get slightly different results of 4379 wins for player 1, 4153 wins for player 2, and the remaining 1468 were draws. This implies that you are more likely to win the game if you subtract rather than add, which seems odd as you would be reducing the number, and the smaller the numbers the higher the density of square numbers. However these results are fairly close to even so it is not too much of a difference for play styles.

6 Conclusions

Although this project has been unsuccessful at finding an explicit solution to *Epstein's put or take a square game* we have found and explored some interesting results. Firstly having created some programs that allow to quickly show all the possible moves for a set number (Section 3.3) and construct a set of layers that will result in a win for a certain number of moves (Section 3.4). We have been able to make some progress into the generation of elements from the groups of primitive elements (Section 4.1) and giving some analysis of layer 2. We have also seen some presence of primitive elements in the third layer (Section 4.2). Finally by looking at other types of investigation we yield some other interesting patterns that may prove useful in further attempts at the problem (Section 5).

Overall the programs developed for this project we fairly effective at identifying numbers with certain properties fairly effectively. However if given more time it would be worth improving the efficiency of them as it takes quite a bit of time to run for large ranges or for lots of iterations. It would also be a ideal to improve the clarity of results obtained from the outputs. The issue is once the numbers are found they still need to be processed, however due to time constraints it was not possible to add the necessary programs to all the files for optimal outputs.

The results obtained for the second layer regarding the primitive elements is also interesting.We have found that these elements are a subset of two other sets, the solutions to Pell's equation and sum of two squares. This can used to narrow our searching range and perhaps find more properties of these numbers to find all the sets that these elements are members of, and perhaps use the intersection between these groups to identify all elements with these properties.

For the results in the third layer it would have been good to look more in depth at the generation of these elements through primitive elements. It does seem that there are some primitive elements in this set, but they have not been investigated fully.

Further investigations that could have been further investigated in this project would be alterations to the game, instead of using the largest perfect square smaller than our number we could consider prime numbers, powers of 3, or factorial numbers. As some of these options have been explored in other material and have shown to yield some interesting results [1].

A final consideration for this project would have been to look at some of the numbers in

the D_∞ set, and see how a game originating from a number like this would play out for a large number of moves. However attempts at doing this currently were limited by the efficiency and speed of our programmes. In conclusion this project deals looks at some analysis of *Epstein's put or take a square game* and finds some interesting patterns in the initial turns of the game, and although there is not obvious solution at the moment the results found show that the key numbers that form part of the solution are not random and do form some sort of patterns.

References

[1] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy *Winning ways for your mathematical plays, Volume 2* Academic Press Inc, London 1982

[2] Jean-Yves Tinevez *Tree data structure as a MATLAB class*
`https://uk.mathworks.com/matlabcentral/fileexchange/35623-tree-data-structure-as-a-matlab-class`

[3] Dusan Djukic *Pell's Equation*
`http://www-bcf.usc.edu/ lototsky/PiMuEp/Pell-IMO.pdf`

[4] Eric Weisstein *Pell's Equation*
`http://mathworld.wolfram.com/PellEquation.html`

[5] Dickson, Leonard E. *History of the theory of numbers, Volume 2* Washington Carnegie Institution of Washington 1919

A List of all numbers under 10,000 for the first 6 layers

Layer 1: 1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,529,576,625,676,729,784,841,900,961,1024,1089,1156,1225,1296,1369,1444,1521,1600,1681,1764,1849,1936,2025,2116,2209,2304,2401,2500,2601,2704,2809,2916,3025,3136,3249,3364,3481,3600,3721,3844,3969,4096,4225,4356,4489,4624,4761,4900,5041,5184,5329,5476,5625,5776,5929,6084,6241,6400,6561,6724,6889,7056,7225,7396,7569,7744,7921,8100,8281,8464,8649,8836,9025,9216,9409,9604,9801,10000.

Layer 2: 5,20,45,80,145,580,949,1305,1649,2320, 3625,4901,5220,7105,9280.

Layer 3: 11,14,21,30,41,44,54,69,86,105,120,126,141,149,164,174,189,201,216,230,245,261,276,291,294,309,329,344,366,381,405,420,446,461,489,504,534,549,574,581,596,621,630,645,670,681,696,721,734,749,774,789,804,829,846,861,886,905,920,945,966,981,1006,1029,1044,1069,1094,1109,1134,1161,1164,1176,1201,1230,1245,1270,1301,1316,1341,1374,1389,1414,1449,1464,1489,1526,1541,1566,1605,1620,1645,1680,1686,1701,1726,1761,1769,1784,1809,1844,1854,1869,1894,1929,1941,1956,1981,2016,2030,2045,2070,2105,2121,2136,2161,2196,2214,2229,2254,2289,2309,2324,2349,2384,2406,2421,2446,2481,2505,2520,2545,2580,2606,2619,2621,2646,2681,2709,2724,2749,2784,2814,2829,2854,2889,2921,2936,2961,2996,3030,3045,3070,3105,3141,3156,3181,3216,3254,3269,3294,3329,3369,3384,3409,3444,3486,3501,3526,3561,3605,3620,3645,3680,3726,3741,3766,3801,3849,3864,3889,3924,3974,3989,4014,4049,4101,4116,4141,4176,4230,4245,4270,4305,4361,4376,4401,4436,4494,4509,4534,4569,4629,4644,4656,4669,4704,4766,4781,4806,4841,4905,4920,4945,4980,5046,5061,5086,5121,5189,5204,5229,5264,5334,5349,5374,5409,5474,5481,5496,5521,5556,5621,5630,5645,5670,5705,5770,5781,5796,5821,5856,5921,5934,5949,5974,6009,6074,6089,6104,6129,6164,6229,6246,6261,6286,6321,6386,6405,6420,6445,6480,6545,6566,6581,6606,6641,6706,6729,6744,6769,6804,6869,6894,6909,6934,6969,7034,7061,7076,7101,7136,7201,7230,7245,7270,7275,7305,7370,7401,7416,7441,7476,7541,7574,7589,7614,7649,7714,7749,7764,7789,7824,7889,7926,7941,7966,8001,8066,8105,8120,8145,8180,8245,8286,8301,8326,8361,8426,8469,8484,8509,8544,8609,8654,8669,8694,8729,8794,8841,8856,8881,8916,8981,9030,9045,9070,9105,9170,9221,9236,9261,9296,9361,9414,9429,9454,9489,9554,9609,9624,9649,9684,9749,9803,9806,9821,9846,9881,9946.

Layer 4: 29,101,116,135,165,236,404,445,540,565,585,845,885,909,944,954,975,1125,1310,1350,1380,1445,1616,1654,1669,2325,2340,2405,2541,2586,2705,3079,3150,3185,3365,3380,3405,3601,3630,3705,4239,4921,4981,5225,5265,5485,6414,6449,7110,7125,7255,7580,7605,7975,8109,8775,9285,9360.

Layer 5: 52,71,84,208,254,284,285,296,318,353,390,429,444,468,470,491,513,558,605,654,704,
705,758,813,832,870,929,990,1053,1118,1169,1184,1185,1254,1325,1398,1473,1550,1558,1629,
1684,1699,1710,1724,1793,1837,1856,1878,1965,2054,2123,2145,2238,2333,2430,2529,2624,2630,
2664,2702,2733,2781,2805,2838,2910,2945,3017,3054,3126,3165,3237,3278,3350,3393,3465,3480,
3510,3582,3597,3629,3655,3701,3716,3750,3822,3837,3873,3945,3960,3998,4006,4070,4085,4125,
4197,4212,4254,4326,4341,4385,4419,4457,4472,4518,4590,4605,4653,4661,4725,4736,4740,4759,
4790,4862,4877,4896,4929,5001,5016,5035,5070,5123,5128,5142,5157,5176,5213,5285,5300,5319,
5358,5430,5445,5464,5505,5577,5592,5611,5654,5726,5741,5760,5805,5877,5892,5911,5958,6030,
6045,6064,6113,6157,6185,6200,6219,6232,6270,6324,6342,6357,6376,6429,6501,6516,6535,6590,
6662,6677,6696,6736,6753,6796,6825,6840,6859,6918,6990,7005,7024,7054,7085,7157,7172,7191,
7221,7254,7280,7295,7326,7341,7360,7390,7425,7497,7512,7531,7561,7598,7670,7685,7704,7734,
7773,7845,7860,7879,7909,7950,8022,8037,8056,8086,8129,8201,8216,8235,8265,8310,8382,8397,
8416,8446,8493,8565,8580,8599,8629,8637,8678,8750,8765,8784,8814,8864,8865,8937,8952,8971,
9001,9054,9126,9141,9160,9190,9245,9317,9332,9351,9381,9438,9510,9525,9544,9574,9633,9705,
9720,9739,9769,9808,9830,9848,9902,9917,9936,9948,9966.
Layer 6: 257,397,629,836,1177,1440,1818,1833,1901,1937,1988,2210,2263,2280,2501,2516,2612,
2845,2861,3039,3188,3389,3621,3654,3860,4053,4105,4541,4693,4708,4813,4930,5381,5415,5746,
5917,5981,6245,6452,6570,7410,7517,7640,7695,7748,8090,8130,8164,8365,8465,8516,9066,9096,
9300,9461,9495,9605.

B Drawing positions for numbers from 1 to 100 that would result in a draw within the first 10 moves

2,3,8,10,13,17,26,29,34,37,40,50,51,53,58,65,68,73,82,85,88,90,97,101,104,109,116,122,125,130,
135,137,148,153,160,170,173,178,185,192,194,197,200,204,205,212,221,226,229,234,241,250,
257,260,265,272,273,281,290,293,298,305,314,325,328,333,340,349,352,360,362,365,368,370,377,
386,397,401,404,409,416,425,436,442,445,450,457,459,466,477,485,488,493,500,509,520,530,533,
538,545,554,560,565,577,585,592,601,612,626,629,634,641,650,661,671,674,677,680,685,692,693,
701,712,725,730,733,738,745,754,765,778,785,788,793,800,809,816,820,833,842,845,850,857,866,
877,890,901,904,909,916,925,936,962,965,970,975,977,986,997.