International Journal of Research in Circuits, Devices and Systems

E-ISSN: 2708-454X
P-ISSN: 2708-4531
IJRCDS 2021; 2(2): 55-59
© 2021 IJRCDS
www.circuitsjournal.com
Received: 09-05-2021
Accepted: 12-06-2021

Roshan Ahmed
Vellore Institute of Technology, Chennai, Tamil Nadu, India

Correspondence
Roshan Ahmed
Vellore Institute of Technology, Chennai, Tamil Nadu, India

Modeling snakes and ladders as an abstract machine

Roshan Ahmed

Abstract
On comparison from a certain angle, the popular classical board game, Snakes and Ladders, is similar in format to string computational devices, leading to an interesting relationship between them. Thus, this paper will be detailing on how to model an abstract device that uses the computation pattern similar to snake and ladders, with N states and a random number from 1 to n jumps for each character, including the "snake" and the "ladder" which will put the current state to a different state in forward or in the backward directions. We will mainly, analyze the time to "finish the game" also known as the total computation cycles by running simulations of it with different values for n and N, with randomized and special mathematical pattern generated snakes and ladders. The benefits of developing such an abstract device would be beneficial on the fronts of privatized, yet mass applicable many-to-one checksum method. This checksum method will work by determining if the letters of a public encrypted string, if converted to dice rolls, will be able to “win” in a private game board of snake and ladders, thus additionally, a method to generate such abstract game boards to create discreet checksums for different purposes, such that if a string is accepted on one board, it should not be able to win on another.

Keywords: Checksum, snake and ladders, abstract machines, state absorbing markov chains

1. Introduction
The classical game Snake and Ladders definitely is a game with a generally long history and many variants, which is fairly significant. A game called “Leela” is one of the first recorded variants of snake and ladders where the player gain either “Knowledge” or “injury” which particularly makes you the player go up or down also known as the snake or the ladder. Snakes and Ladder game has 72 states, where the 68th state basically is the winning state [1].
The Snakes and Ladder game was mostly that kind of game which was designed to educate people about the morality and basic moral values. In this game each state is represented by a different section of moral value. The “snakes” and “ladders” represented different kinds of actions instead of states. In general each action leads to its equivalent reward (knowledge) or punishment (injury). For example every reward (knowledge) takes you to a higher state via the ladder, whereas every punishment (injury) sends you to a lower state via the snakes and sometimes the reward or punishment would impact in a very significant way.
With the passage of time, the intent with which this classical and educative game was created lost its primary objective. People no longer looked at this classical game for educational purposes anymore and it pretty much became a fun game for friends and families. Similar is the fate of such games, for example Chutes and ladders, the early American Variant that had cartoon illustrations representing different habits, both good and bad, that affected pretty much how the player would progress through the game.
Another game, which particularly came much later as an American variant; featured a 7x7 grid representing different states of the United States of America. This mostly was in a way a tourist game, where the player definitely has to travel to each state, and there really were 3 “slides” and 3 “steps” that effectively did the same thing as the other very European version.

1.1 Goals of the paper
This paper will specifically talk about the computational aspects of the classical game snakes and ladders, and what we can represent it as in a computational perspective, which really is quite significant. For all future purposes in this paper, the Snakes & Ladder Machine will be abbreviated to “SNL Machine” for simplicity.
The SNL machine is particularly an abstract machine that defines as a machine with N states, the Nth state being the winning/final state. We will also generally be having “trap” states after Nth state that will not literally be counted as winning states.

These trap states and the different ways to specifically bypass them will really be discussed in detail in basically further sections of this paper in a generally big way. This paper also proposes that the SNL machine can be used to essentially detect a for all intents and purposes specific form of checksum, which basically is fairly significant.
The implications and the benefits of which shall basically be discussed. This paper will also kind of discuss the applications of the SNL machine in security, verification and encryption systems or so they mostly thought.

2. Methodology of research

It is important to first discuss the approach/methodology of research and experimentation. Despite it not being relevant to the topic of the research, the method of simulation of data is important for future replication and verification of said data.
The following research is done on a simplified version of snake and ladders written in C++ TDM-GCC version 4.9.2 64 bit release. Here we have a few important functions to discuss, in this case, namely, the rand function, the time. Header file and the seed generation. The specifics of the seed are given as follows:

$f(x)=1000\ x+((x)/(1000))$

Where x is the system output value for get time of day (&tp, NULL), f(x) is what is known as "time". Each second, the time gets incremented by a specific amount.

$\text{Seed} = (3^{\text{time}}+\text{time})^{\text{time}}$

$\text{Random number} = |\ 372 * \text{seed}^{|73*\text{seed}*(\text{seed}\%15)|}\ |$

The random number is then truncated to the relevant size. Due to the huge size of the said generated number, truncating it provides a wider range for experimentation, thus providing us with a largely varying sample numbers.
This was done mainly to avoid the dangers of RNG generating very similar or in fact, nearly the same numbers. So we have taken that extra step for the purpose of maintaining integrity of experimentation.

3. Why this model?

Before going into the details of the model, we will see the gist of why this model has been proposed.
Firstly, this model proposes an intransitive checksum based verification system.
Secondly, the system given here is hard to replicate or reverse engineer, thus protecting the authenticity of the keys
Thirdly, the given keys used in the model are far smaller than the RSA verification system of prime numbers [9], meaning that encrypting and decrypting is faster, while keeping keyless encrypting at its maximum difficulty.
Fourthly, for an N sized buffer space, there are a total of nearly N!/2 possible private keys. Comparing this to N sized prime number, where only by checking only N/2 numbers, the key can be acquired. For larger numbers, this becomes more effective as the size of the factorial grows larger and larger.

4. First model-no snakes & ladders

We start with N number of states. Our sample machine taken here has 16 states, the 16th state being the finishing state.
We have taken the die to be 4 sided including 0 where the set of possible rolls are {0, 1, 2, 3}. This is made so that we can have a crumb-wise (2 bits) [2] computation. We will also be limiting our input values to 6 crumbs or 12 bits of data and the "win state" being 16.
We have not taken any snakes or ladders for the first run, and we have only taken 12 bits of data.

Table 1: No Snakes or Ladders 6 crumb

End State	Total games	Games Passed	Pass Rate
14	100	6	6%
14	360	20	5.5%
16	360	3	0.83%
16	360	2	0.5%

The low pass rates of 16 as compared to 14 is the original reason why we took 16 as the end state as compared to 14 to reduce the brute force strings of data to be accepted.
For credibility of data, we will be using different bit lengths and end points to see the one with the lowest pass rate.

Table 2: Size and State-pass rates

Crumbs	End State	Total games	Pass Rate
6	16	360	0.5%
7	16	360	3%
8	18	360	3.88%
8	20	720	0.41%

5. Markov Chains and Snakes & Ladders

On analysis by multiple people [3, 4], Snakes and Ladders are a form of self-absorbing Markov chain. From, their data, they have discovered that if you add a "snake" also known as a positive chain to the original chain that skips n steps, the probability of finishing the game increases proportionally ton^2, the number of steps skipped. Similarly a "ladder" also known as a negative chain of size m decreases the probability by a number proportional to m^2. Adding both a snake and a ladder of equal sizes means that we do not significantly change the probabilities of the game, meaning that the above data will remain credible if we have one snake and ladder that are equal and opposite to each other.
There are, of course, exceptions when there are snakes or ladders in vital spots or for example there are too frequent or relatively large, even possibly putting the chances of a game ending in a set number of moves to sub 0.05% values.
Thus in our SNL machine we will be having one snake and ladder at different position and sizes to check the difference in the pass rates.

Table 3: Snakes and Ladders addition

S. No.	Crumbs	End State	Total games	Pass Rate
1.	6	18	360	1.1%
2.	6	18	360	1.3%
3.	6	18	360	1.3%
4.	8	18	360	2.7%

Table 3 explains about the pass rates for different strings have been taken over by different combinations of snakes and ladders. Note, that in each case, the average addition of the state progression by each snake and ladder is equal and opposite.

6. Method of calculation

The data presented in this paper is not calculated in any form. All of it has been experimentally generated in Dev C++ using the sys/time.h extension.

The seed used is generated by T=> time = tp.tv_sec * 1000 + tp.tv_usec/1000;

The next relevant function will be the amplifier. The purpose of this function is to take a small difference and amplify this difference to extreme amounts due to holding a very high differential value.

Seed s is then derived from T using the formula s=((3^time)+time)^time.

And finally, the number itself is generated using = abs(327*rand()^abs(73*rand()*(rand()%15)))%X.

Here, X is the limiter, which limits the number to 6, 7 or 8 crumbs. X= 4,096 for 6, 16,384 for 7, and 65,536 for 8 crumbs respectively.

The number that is generated is taken as "Data appended by Checksum in base 4" for all our calculation where the first 4, 5 or 6 crumbs represent the "Data" for 6, 7 and 8 crumb numbers respectively, unless otherwise specified.

The C++ program moves the token after reading MSB of the 4 bit number, then checking the snake or the ladder condition, then proceeding to the next MSB. The position's final position is the "Finishing position" or "End state" of the string, and it is accepted if and only if the finishing position is in the final state of the machine.

7. Checksum proposal

The checksum proposed will specifically have 3 parts, the data, the Checksum and the internal private key. As most private keys, the kind of key literally is only available to the party that for the most part is authorizing the given data.

In this case the private key that we will be giving will be the data of the snakes and ladders. The snakes and ladders will have the starting and ending states. The overall data for the 6 crumb + 2crumb checksum will be as follows:

CC CC DD DD DD DD DD DD

Meanwhile the private key for 2 snake/ladder combo will be as follows.

SSSS EEEE SSSS EEEE

In this specific example, the bits represented by S determine the starting 4 bit number of the ladder, and E determining the ending 4 bit number of the snake and ladder. Adding further snakes and ladders will comprise of 8 additional bits. If the size of the board increases, the number of bits increased will be 2n where n is the number of snakes and ladders on the board currently.

This, of course actually is a small example of what the checksum would generally be like, which is fairly significant. For particularly practical terms, the magnitude of data could particularly be bigger and so would actually be the checksum, states and number of snakes and ladder in a subtle way.

We have particular goals and targeted purposes for this system, which we will now discuss.

The system that we develop should be a single string processing unit with a private key, such that if we input our string, it should tell us if our string is a valid string in said language.

Furthermore, the private key needs to be distinct, and the system needs to be able to process a large range of strings. This means our system must have 2 main features. The system must be mass applicable (applied to a large amount of strings) and non manipulable (brute forcing and reverse engineering must not be easy).

This puts us at a strange state, where the percentage of strings that pass the checksum must be decently high, but should also neither be too low as to avoid having any freedom of choice and variety, but it should also be low enough to avoid random numbers passing the string test. In most existing RSA [9] verification systems, uses large prime numbers, since there are a lot of them, and yet, are only a small percentage of the existing numbers, that can be classified as prime.

We believe, that with the snake and ladder mechanism, we can overcome a few of those issues.

The benefits of the snake and ladder machine are as follows [5]:

- The checksum is intransitive, meaning 123 may not be the same as 321 for the checksum.
- For an outsider who does not know of this checksum mechanism used, a Snakes and Ladder machine based on Markov chains will be hard to reverse engineer. Only brute force mechanism will work
- The larger the size of the checksum, the longer it will take to brute force it. For larger terms, the reverse engineering to find the private key becomes nearly impossible, mainly due to lack of transitivity
- If a prime number is used to generate the snake and ladder key, combined with any other encryption method, the original check can only be retrieved by the original snake and ladder board.
- The Snakes and Ladder machine can be given the data, and it will return the checksum. This checksum,. Due to the pigeonhole theorem [6], will be there for multiple different sets of data. In this case, it can be used to our advantage to avoid brute forcing the board to find the private key.

At the same time, disadvantages of using this Snakes and Ladder machine for checksum would be:

- Multiple boards may return the same checksum, and at the same time, a checksum can work for fairly multiple data (pigeonhole theorem) for which, of now, there is no solution, which basically is fairly significant.
- This makes collision happen which lowers the strength of the checksum, despite being hard to reverse engineer, the actual checksum might collide.
- It specifically is possible to have data which cannot literally be made into checksum, which is quite significant.

8. Discreet finite automata for SNL Machine

The finite automata of the machine is as follows. On input of 0, it goes to itself. On any non-zero input, it moves to the state 1, 2 or 3 states in front of it. If the destination state has a snake or ladder, it instead moves to the final end of the snake or ladder.

The machine on viewing, feels like it is a NFA (Non deterministic finite automata) which can be converted to NFA. The main cause of this is the states after the finish state, which in this paper; we will refer to as endgame states. If not taken care of, these states will have an overflow. To avoid this, we have a few possible solutions.

A. Loop around: We can make the board circular in nature. With the N+1th state equaling 1st state.

B. Endgame states compatibility: We can have states beyond the Finish state, then one Snake at the end of the overflow. This is similar to the rules in Leela [1], where you have a snake from the end to another state in the middle of the board. This means there is a partial looping, which is not as big as having a full on circular board

C. Endgame Rejection: We can reject any string that reaches the endgame states. The downside to this is that there are some strings that cannot be processed correctly if you discard the endgame strings.

D. Signed checksum endgame compatibility: Here we have the states flow for a certain distance above the finish state, but we keep the checksum signed. This means if the final state after processing the string is as E+x, where E is the first end game state, then the checksum value can be-x. which lets the endgame strings be processed correctly. The downside to this will be starvation of lower level final states that may need larger checksum values to reach the finishing state.

Once any of these methods of handling overflow is implemented, it becomes a simple DFA. Since we need to develop a checksum, we need to first be able to check if a state can reach the final state just by adding the checksum at the end. If it can, then we will provide the checksum based upon the final state it ends on.

Let's take our example Snakes and Ladder machine with 16 states and 6 data crumbs with 2 crumbs for checksum. We will have only one snake, going from 7 -> 2 and one ladder, going from 9 -> 14. We have chosen to reject the states beyond the finishing state. The DFA for end state will be as follows.

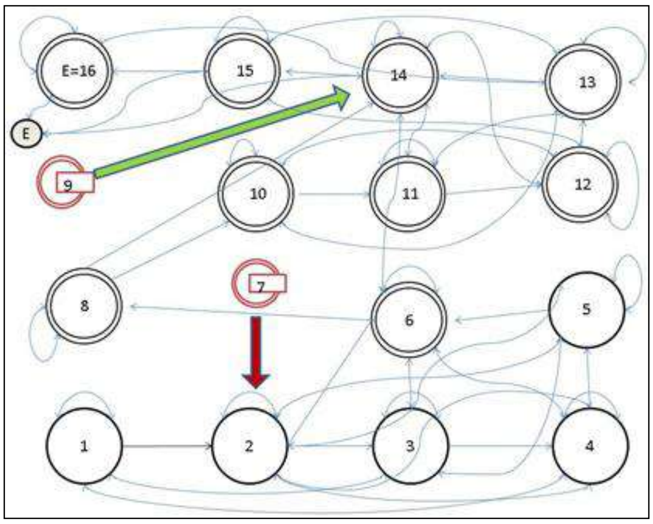

Fig 1: DFA of the Snakes and Ladder machine

Fig. 1 explain about total DFA of the Snakes and Ladder machine. In green, the ladder and in red, the snake. We can see that we can simplify the DFA to remove state 9 and 7, since there is no way to reach them, since the tokens that land there are directly transferred to the end of the path.

This diagram represents the DFA for all strings of any size that at the end of which, can be assigned a checksum of 2 crumbs. We can see that any state above 17 and below 5 cannot be assigned checksums. Only 47.08%, out of experimental analysis, were accepted into states that allowed a valid checksum.

9. Why Snakes and Ladders

This probably brings up the question of why we chose snakes and ladders from amongst the many board games available, and why we chose such a topic. Let us review a few commonly available board games and how they would fare as a machine.

Chess: Chess is an open ended, one to many format of a game. If we input each "move" as a "string" into a machine, there would be multiple possible issues. Being open ended, it is possible to enter possible moves that have nothing to do with the string that is entered. For example, assume we started from a standard chess board, the "fool's mate" [7], which can be paraphrased into the string:

1. f3 e6
2. g4 Qh4#

The above move string is the shortest possible checkmate sequence. That means we can simply just enter this string, and it will be verified as a valid string, thus the system being broken before it can even be expanded further.

Moreover, chess, being an open ended game, determining a "win state" is difficult from a given set of strings. One potential work around is to choose a player and a particular starting board, which can be saved as a private key, and the input strings will be from there, rather than compared to a standard starting board.

Ludo: On a first look, the game, ludo, is really similar to snake and ladders. The only difference being that instead of having snakes and ladders, we have other players that determine our movement. There are precisely 3 reasons why ludo may not function well as an abstract machine.

The first reason is because ludo is not a linear game.it requires control of multiple players and multiple tokens to play. Thus the standard board of ludo will have a total of 16 tokens to play with. The chances of one player winning is nearly ¼, for a given string. This level of complexity is not optimal, and is unusable in terms of efficiency. It might work for longer strings, in fact, but the chances of it functioning need to be determined further.

Poker: Poker is a well-researched game due to its popularity in terms of gambling and capitalism. If we input the count of cards and their face values as strings, then we can create yet another abstract machine. The issue with this, similar to chess and ludo, is that this game is not linear, and the performance of one players, also known as one hand, depends entirely on the hand of another player. If we decided to create a private key as a specified hand, and the string input as another hand, and the "win state" as the state hand where you can beat the private key's hand, in this case, you can input the "royal flush" or a "one of a kind" hand [8], you can beat the hand, despite it being anything. This poker based checksum system again beats the purpose of a verification system.

Uno: Another popular choice of a card game, where once again, the play style is similar, but the processing of the cards is done in a different system. Here we can see if our hand can be completely finished by the end of the string, or if we still have cards left over. This sort of system might seem to work, but the main problem with Uno is that the current hand that can be played is heavily determined by the previous card. The system is synchronous, where the string character depends on the string before that. This also means that we can, in some form, determine the entire hand simply by looking at the starting card. Despite being long, it is entirely possible.

Moreover, the system being synchronous means that there is a serious limit on the list of strings that we can encrypt. For example, in our snakes and ladders machine, we determined that only 47.08% [refer section V] of the strings we have can form a checksum, for Uno, it is about 0.000303% of the series of cards, ignoring the wild and power cards (plus 2, plus 4, reverse and skip) for a base string of 6 cards. This low freedom rate makes it so that only a very minor amount of strings can be processed, thus is in effective for mass privatized checksum. It may work for a single one-to-one verification, but a privatized mass system may not be functionally efficient.

10. Expansion of SNL machine

The Snakes and Ladder machine can be expanded infinitely. For every crumb added, the final state states must be moved by +2.

Or if N is the number of states/the end state, and b is size of data checksum in bits, then N= 4+b where b is at least 4.

The number of snakes and ladders have not been experimented with so far, but we can estimate one pair of snake and ladder every 10 or so states after a minimum of 20 states.

In the last, new innovation with emerging technologies (post COVID 19) can be found in [10-15]. These articles will help future researchers to find their research problem for continuing their research work.

11. Conclusion

This way we for all intents and purposes have designed a generally simple kind of abstract machine based on snakes and ladders, including the potential checksum that for all intents and purposes comes from it. It is in no way perfect or complete and kind of has a lot of flaws in it. Future fixes and research into this model of checksum machines for encryptions might help to essentially fix a lot of its flaws to perfect it in a major way.

It kind of has the generally potential to mostly be a different kind of checksum that isn't purely kind of dependent on particularly prime numbers or the kind that can actually be easily altered by exchanging digits, which is fairly significant.

12. References

1. Harish Johari, Leela. The Game of Self-Knowledge: Commentaries, 1975. ISBN 0-89281-419-5.
2. Weisstein Eric W. "Crumb". Math World. Retrieved 2015-08-02.
3. "How long is a game of snakes and ladders?" The Mathematical Gazette. 993 March;77(478):71-76.
4. Markov Chains, Snakes, Ladders DE, Daykin JE Jeacocke, Neal DG. Data Integrity Checksums, Meghan McClelland, 2018.
5. Pigeon hole principal WA Trybulec-Journal of Formalized Mathematics, 1990. 212.33.73.131.
6. Beale, (.pdf p. 49), 1656, 17.
7. Greiner Ron. The Everyday Guide to Recreational Poker. Everyday Endeavors, LLC, 2005, 46-60. ISBN 0-9769703-0-9.
8. The Original RSA Patent as filed with the U.S. Patent Office by Rivest; Ronald L. (Belmont, MA), Shamir; Adi (Cambridge, MA), Adleman; Leonard M. (Arlington, MA), U.S. Patent. 1977 Dec;4:405-829.
9. Nair MM, Tyagi AK, Sreenath N. The Future with Industry 4.0 at the Core of Society 5.0: Open Issues, Future Opportunities and Challenges, International Conference on Computer Communication and Informatics (ICCCI), 2021, 1-7. Doi: 10.1109/ICCCI50826.2021.9402498.
10. Tyagi AK, Fernandez TF, Mishra S, Kumari S. Intelligent Automation Systems at the Core of Industry 4.0. In: Abraham A., Piuri V., Gandhi N., Siarry P., Kaklauskas A., Madureira A. (eds) Intelligent Systems Design and Applications. ISDA 2020. Advances in Intelligent Systems and Computing, Springer, Cham., 2021, 13-51. https://doi.org/10.1007/978-3-030-71187-0_1
11. Varsha R, Nair SM, Tyagi AK, Aswathy SU, Radha Krishnan R. The Future with Advanced Analytics: A Sequential Analysis of the Disruptive Technology's Scope. In: Abraham A, Hanne T, Castillo O, Gandhi N, Nogueira Rios T, Hong TP. (eds.) Hybrid Intelligent Systems. HIS 2020. Advances in Intelligent Systems and Computing, Springer, Cham., 2021, 13-75. https://doi.org/10.1007/978-3-030-73050-5_56.
12. Tyagi Amit Kumar, Nair Meghna Manoj, Niladhuri Sreenath, Abraham Ajith. "Security, Privacy Research issues in Various Computing Platforms: A Survey and the Road Ahead", Journal of Information Assurance & Security. 2020;15(1):1-16.
13. Madhav AVS, Tyagi AK. The World with Future Technologies (Post-COVID-19): Open Issues, Challenges and the Road Ahead. In: Tyagi AK, Abraham A, Kaklauskas A. (eds.) Intelligent Interactive Multimedia Systems for e-Healthcare Applications. Springer, Singapore, 2022. https://doi.org/10.1007/978-981-16-6542-4_22
14. Amit Kumar Tyagi, Aghila G. A Wide Scale Survey on Botnet, International Journal of Computer Applications. 2011 Nov;34(9):9-22. (ISSN: 0975-8887).