Game level generation with recurrent neural networks

Banno Postma
10444602

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. E. Gavves

QUVA Lab
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 24th, 2016

1 Abstract

Procedural content generation is to create game content with limited to no human interaction. A general problem is creating diversity in the generated content and making it believable in the sense that it gives the player the impression that a human made it. In the case of game levels it would be useful if existing levels can be used as a base to generate more levels. Recurrent neural networks have had much recent success with handling sequential data in classification problems such as text prediction. In the case of a linear level a recurrent neural network can predict new data from a training set such as existing levels. In this paper a recurrent neural network model is created to generate Super Mario bros levels. By fitting the model to the original levels it becomes able to return a set of possibilities for next steps in the sequence. A new level is generating by sampling from this set of possibilities. The believability and diversity of these levels are measured with a set of metrics and a human feedback experiment. The system is becomes able to generate levels with sufficient believability and diversity however it is not able to do this consistently.

Contents

2 Introduction

Procedural content generation or PCG refers to algorithmic generation of game content such as levels, textures or characters with limited or no human contribution. Procedural content has been used in games from the early eighties. In 1980 PCG was used in the game Rogue to generate dungeons at runtime. A more recent example is Minecraft, a popular sandbox game where an entire world complete with trees, caves, animals, monsters and structures is procedurally generated. Currently PCG is also widely used in mobile games to create infinite levels. The goal of these levels range from achieving high-scores to simply the joy of being able to infinitely play the game without having to replay the same levels.

A general problem with level generation is creating diversity. It is important that there is some diversity inside the levels themselves and between the different generated levels. A level where the player needs to jump over the same obstacle a hundred times is quite boring. Similar to this is a game where the only difference between levels is the color palette . The other extreme is a level that is completely randomly generated. This level would probably be unplayable and senseless. There is a need for balance between these two extremes.

Another goal of level generation is achieving some creativity and believability. The player should not feel like the level has been generated by a procedural content generator. Not many people would believe that the randomly generated level from previous example was designed by a human. A sense of purpose and structure is important for believability.

To achieve these goals there are multiple solutions that range from generation grammars and rule sets to premade patterns that are controlled by a range of variables. An apparent weakness is that these grammars or patterns need to be designed beforehand which takes time and is prone to errors. A different approach could be generating the new levels out of already existing levels. A possible way to achieve this is with use of deep learning techniques such as recurrent neural networks.

This leads to the question can deep learning techniques be used to improve procedural content generation of game levels?. There is definitely potential seen from recurrent neural network based text generators that already are able to generate coherent sounding text. With use of deep learning techniques the diversity and believability of human created levels could be achieved by the generator. This of course is a broad question since many deep learning techniques can deployed. This project will focus on the use of recurrent neural networks and the following research question:

Can a recurrent neural network while trained on the original levels of a game generate new levels with the same believability and diversity?

This question will be addressed by generating levels for the game Super Mario Bros from 1985. A neural network will be trained on the original levels of this game. After training the network should be able to generate new Super Mario levels. By comparing the generated levels to the original levels the performance of the generator can be measured. The goal is to generate levels with as much diversity and believability as the original levels but also that there is enough diversity between the generated levels.

The expected results are that as the model is fit to the training data it should be able to generate similar content with the same believability. As the training set becomes larger more different combinations between levels become possible allowing the model to generate levels with the same or even more diversity.

3 Literature review

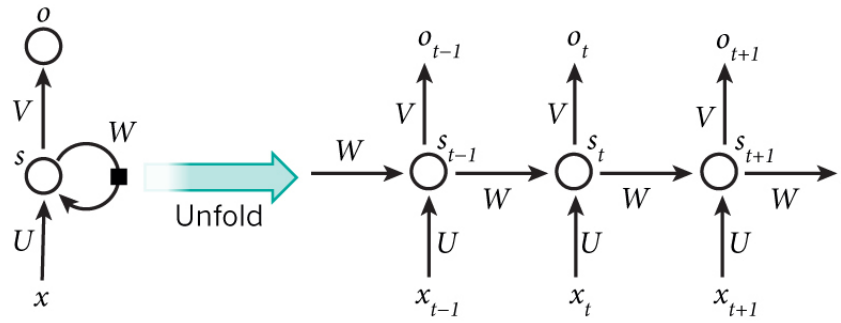
According to Togelius [3] there is a fine balance in procedural level generation between using repetitive structures and a sense of purpose for the player. When the generated level seems too random to a player it loses it sense of purpose but when the structures are too repetitive it becomes boring. The generated levels need to provide the same 'feel' to a player as a human created level would but also provide a substantial challenge for the player to complete. Dahlskog, Togelius and Nelson [2] investigated the possibilities of using existing levels as a base for level generation by creating a level generator that made use of n-grams. They broke down Super Mario levels into sequential columns containing 15 blocks each and acquired the n-grams of column sequences up to $n = 3$. By making use of the probabilities of the n-grams they were able to generate new column sequences. They acquired the n-grams of all the column sequences in the original Super Mario Bros. levels and generated levels using the associated probabilities. They found that the resulting levels were similar in visual appeal and playability as the levels used to create the model. When they expanded the amount of levels used to learn the model the variety increased together with surprising shifts in style.

With the rise of deep learning in AI it is interesting to investigate how this can be applied to the problem of procedural content generation. Recurrent neural networks (RNN) as described in [1](Chapter 10) have been proved to provide excellent results when predicting sequential data. Especially the long short-term memory RNN architecture has produced one of the best known results in handwriting and speech recognition. Making use of a LSTM RNN to generate Super Mario levels can be described as a variation of the n-gram technique as described above. The difference being that with n-grams the vocabulary of columns is limited to the columns present in the original levels while LSTM RRN's have the possibility to use character based generation or a intermediate representation of columns.

A difficult aspect of procedural level generation is automatically evaluating the generated levels. There are two main reasons that make evaluation important. In the first place without automatic evaluation the only way of knowing that the generator performs well is to actually play the levels which is highly inefficient. Finally integrating evaluation into the generation process can highly improve performance by preemptively eliminating bad generation cycles. This project will make use of evaluation as described in [4](Chapter 12) and the paper [5].

3.1 Recurrent Neural Networks

Recurrent neural networks or RNNs are a form of neural networks for processing sequential data. In traditional neural networks it is assumed that the inputs and outputs are independent of each other. When working with sequential data there is a need for 'memory' of some sort where the information of the already processed parts of the sequence is stored. In RNNs the same calculation is performed for each part of the sequence where the input to the recurrent network is not only the observation x_t but also a memory state s_t computed from the previous time steps in the sequence. Figure 1 shows how a recurrent network can be unfolded into a regular network with a repetitive structure.



Where x_t is the input at time t. s_t the hidden state at t and o_t is the output at t.
Image source: Nature

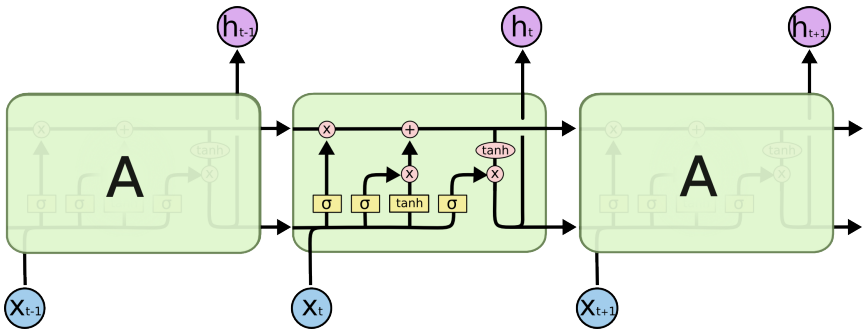
Figure 1: Unfolded recurrent neural network

The time variable t does not need to refer to actual time but more to the position in the sequence. In theory RNN can operate on a sequence with t ranging from 1 to any natural number. In practise the network operates on minibatches of such sequences.

3.1.1 LSTM

RRN use the the previous information of a sequence on the current calculation. This works well when the relevant information is close, for example predicting the next word in ”The color of snow is ...”. The recent information from the words ”color” and ”snow” makes it clear what word is expected. Cases where this information is further back in the sequence called long-term dependencies are troublesome. In theory classic RNNs are capable of learning these dependencies however in practice they do not seem to be able to handle them. This is explored in [6].

The most commonly used RNN called LSTMs for Long short term memory solves the problem of long-term dependencies. As they are explicitly designed to avoid this problem. LSTMs have roughly the same structure as a regular RNN but the repeating inner layer cells are upgraded with additional inner layers. These layers determine what information is forgotten, what information is added and what the output is for the next cell. This way the information of important parts of the sequence are remembered and less important information is forgotten.



Where the cells upper input is the previous cellstate c_{t-1} and bottom input is memory M_{t-1} and observation x_t. Outputs c_t and M_t.

Figure 2: graph of inner LSTM structure. Source: "Understanding LSTM networks"

The first σ layer outputs a number from 0 to 1 deciding how much to remember. Where 0 is forget everything and 1 remember everything. The second σ and tanh layer decide what new information will be added by respectively modulating the input and generating candidate memories. The final σ layer modulates the output and together with the tanh gate over c_t generates a new memory. By fitting the weights of these four layers the model becomes able to learn long-term dependencies.

4 Research Method

Necessary for this project is a target game which original levels can be used as a dataset, a neural network, algorithms that can evaluate generated levels and a method to play the generated levels. The target game for which levels will be generated is the game Super Mario bros. The Neural network will be created using Python and the library Keras. The evaluation methods will be writen in python as well. Finally a python implementation of Super Mario bros. will be used to enable playing generated levels.

4.1 Super Mario Bros.

Originally released for the NES in 1985. Super Mario bros. is a side-scrolling platform game were the player controls Mario and needs to reach the end of the level while avoiding monsters and jumping over gaps. The levels are linear and are transversed by Mario from left to right. Most objects in the game world are blocks of the same size. This size will be 1x1. There are some exceptions such as pipes or moving platform that are 2x2+n and 2x1 respectively. Mario himself is 1x1 normally and 1x2 when powered up. Levels themselves average around 200 by 13. This makes the levels a set of 200 sequential columns of length 13. The sequential nature of the levels allows them to be used in a sequential model such as a recurrent neural network.

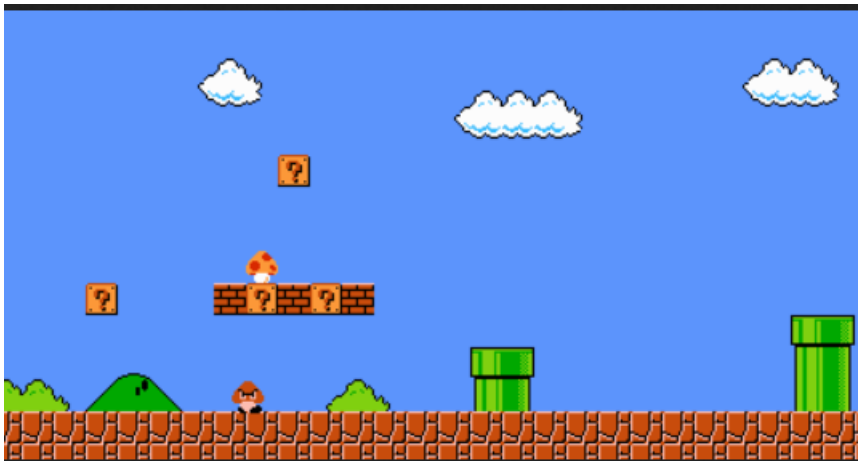


Figure 3: A basic Super Mario bros. level

There are different types of blocks in a Mario level. The basic types are:

Non-solid
This includes air and coins. These blocks have no effect on Mario's movement. The coin gives the player a higher score when picked up.

Solids
Include ground, pipes, coinblocks or powerup blocks and breakables. Mario can not pass through these blocks making standing on these blocks the primary way to navigate a level. Some solids have a special effect when hit from below. Coin blocks give the player a coin, powerup blocks give a powerup and breakables are destroyed.

Monsters
Monster blocks can more accurately be described as monster spawn points and can be regarded as non-solid. When the level starts a specific type of monster will start on the location of the monster block and assume the movement pattern associated to this type of monster. When a monster hits Mario in his normal form the player loses a life and starts the level from the start. When hit while powered up Mario returns to normal. Monsters are generally defeated by jumping on top of them.

Moving
Finally there are moving platforms. Platforms will move vertically or horizontally and continue to do so back and forth for a set distance. Besides moving they behave like solids. When Mario becomes stuck between a moving platform and another solid it will cause the player to lose a life. Moving platforms are not designed to collide with other solids. A collision of such would be a generation error.

All these different types of blocks make up a level. This information needs to be in an appropriate format to be used by the neural network. Text seems to be fulfilling this need. Every different block encodes to a different character. A disadvantage is that that a pipe which is a 2x2+n structure needs to divided into different pieces. Figure 4 shows the results of encoding a Super Mario level. Notice that one line of text does not give much information about the level opposed to the information gained from looking at each column. For convenience the text file used in the neural network will be flipped in such a way that every line represents a column of the original level. This is shown in figure 5. This way all original outside levels from super Mario are encoded and used as data set. In the game there are outside and inside levels. The inside levels are not used in the data set because there is too much of a stylistic difference between the two level types. For example the presence of a roof in all the inside levels.

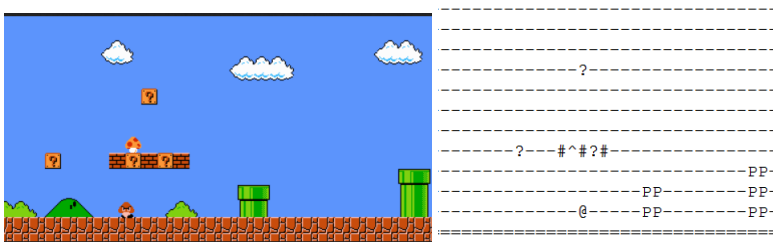


Figure 4: From original level to text

```
-------?---=
----------=
----------=
----------=
-------#---=
-------^---=
---?---#--@=
-------?---=
-------#---=
----------=
----------=
----------=
```

Figure 5: Final form of input

4.2 Generation with Keras

The neural network is created with Keras. Keras is a neural network library for python build on top of Theano. Theano is a python library that allows fast calculation of mathematical expressions by making use of the GPU especially ones with multidimensional arrays. The level generator will be a python script that takes the text file of original levels as input and outputs a text file with a variable amount of levels.

4.2.1 Input

The input is text file of the original Mario levels. There are two different sequences that can be derived from this file. First is the character sequence. This sequence simply consist of all the characters in the file including the new line character. The new line character states when a new column starts. The other sequence is the column sequence. Instead of looking at every single character the input is split at the new line characters, the result is a list that is a sequence of all the columns in the text file. This means that the levels generated from column subsequences are limited to only contain columns found in the dataset. The system will work with columns or characters as tokens.

In both cases the input is processed the same way. A set is created of all the different tokens. By enumerating this set two dictionaries are created that translate index to token and token to index. The next step is cutting the input into smaller sequences of variable length. Two parameters are involved here. First there is the length of subsequences. Secondly there is the step between the start of each subsequences. For example the string ”abcdefg” with length=3 and step=2 will result in ”abc”,”cde” and ”efg”. For every of these subsequences the subsequent token is stored as a label of this subsequence. The final step of preparing the input is vectorization. Vectorization transforms all the input tokens into one hot vectors. These are vectors with length equal to all possible tokens where all values are 0 except the one value associated to that particular token. The vectorization of the array of subsequences and the array of corresponding subsequent tokens leaves the final input shape. The first will act as the input data, the second as the associated labels.

4.2.2 Model

The model used is a sequential model in Keras with one LSTM layer, one Dense layer and a Activation layer. The Dense layer is used to get the output in the right format which is a vector with the same length as the token set. The softmax Activation layer makes the output add up to 1 so that the output can be interpreted as probabilities. This makes the output a vector with probabilities for every different token in the token set. The model is compiled to make use of ”categorical cross entropy” also know as ”multi class log loss” as a loss function. This loss function is recommended when dealing with classification problems such as predicting the next token in a sequence.

The model is now ready to be fit to the input data. Next to the input data and the corresponding label set there are two additional arguments required to fit the data: the batch size and the number of epochs. The batch size refers to the amount of samples per gradient update. This is used to conserve memory. The number of epochs is the number of times every input sequence is used to update the weights of the network. Since for this project the data set is relatively small a large number of epochs is necessary to optimize results.

4.2.3 Sampling

A model fit to the input data can be used to make a prediction by taking as input a sequence of tokens with the same length as the previously defined subsequences length. The output is the list of probabilities of every different token. To start generating a level a seed is needed to act as the first sequence. To obtain this seed a random section with the desired length from the input data is chosen. This sequence is used as input for the first prediction.

There are several ways to obtain the newly generated token from the set of probabilities. The most simple way is to find the index of the highest probability and get the corresponding token from the index to token dictionary created earlier. However the token with the highest probability is not always the best choice. The highest probability is a safe choice that results in less mistakes but also in less diversity. Another option would be to pick a random token based on the probabilities. To give more control on the sampling process, a diversity parameter and sampling function are introduced. The sampling function makes use of the diversity parameter to either increase or decrease the difference in probabilities of each token by dividing the natural logarithm of the probabilities by the diversity. Diversity = 1 changes nothing, diversity > 1 makes the higher probabilities even more likely to be picked and diversity < 1 levels the probabilities of all tokens.

4.3 Evaluation

It is difficult to evaluate game levels on face value. Evaluation based on user satisfaction is impractical, as computing the gradients during optimization would require constant user interaction. For this project the generator will be evaluated on consistency. Which evaluates the generators ability to consistently generate solvable levels. Generated levels will be compared to the original super Mario levels with a set of metrics that measure level structure and diversity. Finally believability will be tested with human feedback.

4.3.1 Consistency

It is important that a level is solvable. In this case solvable means that Mario is able to reach the end of the level. There are two main obstacles that can prevent this. The first is a gap that is too wide for Mario to jump over. Second is a wall or structure that is too high for Mario to jump over. There is also a combination of the two where a wall that is normally not too high is placed at the end of a gap making progress impossible.

Consistency is measured by looping through the columns of a level and finding all platforms in this column. A platform is a solid with a non-solid above it so that Mario is able stand on top of it. Marios jump height is 4. So from the highest platform in a column the new height Mario can reach is platform height + 4. The system makes use of a simplified way of tracking gravity. When moving at constant forward speed when Mario jumps he will keep rising for 3 columns. Afterwards he will fall one block in the next column, two in the column after and so on. Tracking Marios maximum potential height allows the system to know if Mario can reach the top of a wall or the end of a gap.

4.3.2 Metrics

The following metrics are used to evaluate a level. Leniency, linearity and density give a measure of the levels structure while pattern density, pattern variation and column diversity give a measure of diversity present in the level.

Leniency
This metric attempts to capture the difficulty of a level. To calculate this the sum is taken of values assigned to the following elements, in similar fashion as proposed in [7]

- Gaps: -0.5
- Average gap width: -1
- Regular enemies: -1
- Powerups: +1

Linearity
Captures the height differences of the individual paths through a level. By calculating the difference in height of platforms between subsequent columns. If there are multiple platforms in a column the average height is used.

Density
Captures how many different heights Mario can stand on throughout a level giving an approximation of how many paths there are through a level. This is calculated by the average amount of platforms per column.

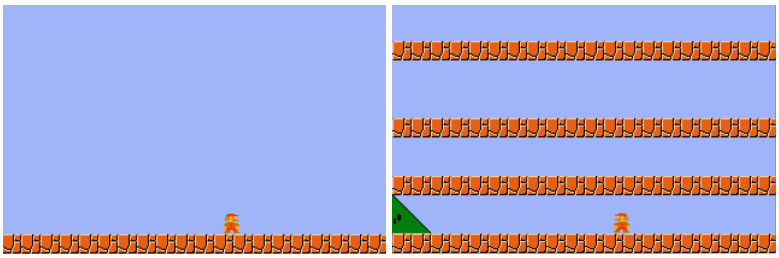


Figure 6: Low and high density respectively

Pattern density
Measures how many patterns from the original Super Mario Bros game can be found in the level. This calculated by the amount of four column blocks from the generated level that are present somewhere in the original levels.

Pattern variation
Measures the unique occurrences of patterns. The opposite of pattern density, calculates the amount of four column blocks in the generated level that are not present in original game.

Column variation
The amount of different columns present in the level.

4.3.3 Human feedback

To test believability a human feedback experiment was conducted. Participants were asked to play four levels and rank them in the following categories: Enjoyment, difficulty and aesthetics. Finally the participants were asked what levels they thought were made by a computer. Before the four levels started a practise level was played. This level was mostly flat and contained a few gaps and monsters to make the participant get used to the controls. For each participant the levels were presented in a random order.

The chosen levels for this experiment included two levels from the original super Mario bros and two generated levels. The first generated level was chosen because the evaluation metrics were close to the averages of the original levels. The second was chosen based on it having a relatively low leniency.

5 Results

The following results are obtained by generating 100 levels with a length of 200 columns for every combination of the following parameters: column or character based generation, number of epochs to train the model, diversity parameter during sampling and subsequence length. The diversity parameter set to increasing means that the temperature is initialized low and is increased when a column equal to the previous was generated. The tables with the actual numbers can be found at the end op the paper.

5.1 Column based generation

Consistency is fairly high in column based generation with a large number of consistency ratings above 85%. Increasing the subsequence length and diversity parameter both seemed to lower consistency. In the case of the diversity parameter this corresponds to the idea of the system taking safer choices at lower values of diversity. A similar effect can be noted for all other metrics. Leniency, linearity and pattern density all become lower when diversity increases. Meaning that with higher diversity there is more height difference, more obstacles and less patterns from the original super Mario bros in the generated levels. With the same increase in diversity pattern variation and column diversity also increase. Meaning more unique patterns and different kind of columns per level. With diversity = 1 most metrics overlap with the values from the original levels while pattern density and pattern diversity are almost equal. This means that the levels have similar structure to the original levels and half of the four block patterns are not found in the original levels. Levels generated with diversity < 1 have low column diversity and high linearity. This means those levels consist mostly of straight lines. The resulting levels are comparable to the low density example from figure 6.

5.2 Character based generation

Compared to column based generation character based generation took significantly more time to train the model and to generate levels. This resulted in the data consisting of only ten generated levels per variable combination. The trends of increasing diversity seems to be the same in character based generation. With the increase of diversity the metrics leniency, linearity and pattern density go down and pattern variation and column diversity go up.

There are two main differences. Firstly the consistency of character based generation is lower and a larger amount of epochs is needed to start getting consistent results. Secondly the maximum of the metrics pattern variation and column diversity is significantly higher compared to column based generation. This is due to the fact that character based generation can generate column that are not in the dataset. A flaw can be detected in subsequence length 52 pattern variation. The maximum pattern variation is greater than 200 while there are only supposed to be that amount of four column patterns in the level. This is caused by the newline character. Character based generation is capable of generating columns of variable length. Notable is that there is not much of a difference in average column diversity except in diversity = 1.2.

5.3 Human feedback

Below the percentages that participants ranked the different levels. Average is a level where all metrics were close to the average of the original levels. The participants were ten students around the ages 21 to 26. They had no earlier experience with Super Mario bros. and had limited to no experience with similar games.

Enjoyable	First	Second	Third	Fourth
Original 1	40	0	50	10
Original 2	10	40	30	20
Average	0	40	20	40
Low Leniency	50	20	0	30

Difficulty	First	Second	Third	Fourth
Original 1	0	10	50	40
Original 2	0	60	0	40
Average	0	30	50	20
Low Leniency	100	0	0	0

Aesthetics	First	Second	Third	Fourth
Original 1	20	20	50	10
Original 2	10	40	30	30
Average	10	20	20	50
Low Leniency	60	20	0	10

Computer Generated	Yes	No
Original 1	50	50
Original 2	50	50
Average	60	40
Low Leniency	30	70

The most notable results are that the level with a low leniency is unanimous voted for as the most difficult level and that the participants were not able to spot what levels were generated. The other results are very varied and point towards the participants not knowing a definite answer. From the participants comments they voted for the difficult level as most enjoyable because they liked a challenge or voted low because this level was frustrating for them. The rating of aesthetics can be discarded since most participants did not have a strong opinion about it. A reason why 70% of the participants did not think the low leniency level was generated was because they thought the level was too difficult to be generated by a computer.

6 Conclusion

Can a recurrent neural network while trained on the original levels of a game generate new levels with the same believability and diversity? A majority of the participants in the human feedback experiment could not find the generated levels. This accounts for believability. Looking at the results of the metrics evaluation many of the structural metrics such as linearity, leniency and density overlapped with original levels while pattern variation was equal or higher than pattern density. Column variation was also at points higher than in the original levels. This accounts for diversity. Combined this states that it is possible for a recurrent neural network trained on original levels to generate new levels with the same believability and diversity.

However this only accounts for a certain amount of the levels. There is a possibility for outliers with sub optimal results. Most of the results are dependent on the diversity parameter. With the best results coming from diversity = 1. This means that the best results are achieved from the basic probabilities of every sequences. The seed that starts of the generation is also randomly decided. This makes the generation process highly dependent of chance and the possibility exist that a insufficient level is generated. Insufficient levels range from inconsistent levels that can not be finished to consistent levels that are not diverse or believable. Without a reliable way to filter out the suitable levels there is a chance that levels of lower quality are used. Creating a complicated filter for this purpose is highly inefficient. With modifications this filter could generate levels of its own making the system unnecessary.

Training a recurrent neural network to generate game levels has potential. A percentage of the generate levels are suitable to be used in the actual game. However as long as the system is inconsistent it will be inefficient to actually use this method in practise. The model or the sampling function need to be improved to a point where no inconsistent levels are generated.

7 Discussion

Character based generation has potential to generate more diverse levels than possible with column based generation. Not exploring the possibilities of character based more is a flaw of this project. The lack of level data makes comparing with column based generation difficult. Having the column size be equal to each other would also help making a more accurate comparison. Now sequence length 54 and 108 translates to column sequence length 4 and 8 respectively.

An apparent weaknesses of this project is the lack of depth of the human review experiment. The number of participants is too low for significant results. The lack of experience of the participants with Super Mario bros works in advantage of believability. It is not a far off assumption that a person more experienced with the game would have more success classifying a human or computer made level.

The model used for training and generation is relatively simple. It would have greatly benefited the research to try what results could be obtained from more advanced models. Would this improve performance or would it not make a difference? These are important questions still to be answered. The whole system in general is too much dependent on the diversity variable and the sample function. These had the most effect on the results obtained and were not explored further. Improvement in any of these three would solidify or falsify the research results.

7.1 Future work

As mentioned above a much more extensive human review experiment can provide more insight in the believability of the generated levels. Different groups with varying level of experience with super Mario will have different opinions about enjoyment, difficulty and aesthetics of a level. More experienced players could be more likely to find which levels were computer generated levels. Another option is making use of a Mario AI instead of human evaluation. The performance of the AI on the generated levels can directly be used as feedback by the model to improve generation.

A weakness of column based generation is the inability to generate columns that are not existent in the dataset. By making use of some sort of intermediate representation of columns and a encoder/decoder this weakness could be solved. There is a possibility that by making use of this method not earlier seen columns will be generated.

There are two ways to improve the system as a whole. First is improving the model. This can be done by experimenting with different configuration of more neuron layers but another possibility is improving the loss function. By integrating evaluation metrics into the loss function itself. This will make the model more controllable and in the case that the consistency function is integrated less likely to make inconsistent levels. By making the generator more controllable experiments with different parameters can result in configurations that yield more interesting levels.

Secondly the sample function can be improved. In its current form the systems results are highly dependent on the sample function. By using the same principle as described for improving the model. The sample function can also become more controllable. A technique such as beam search can already be used to sample so that the outcome fits to certain parameters.

Finally there is a more extreme way of trying to use deep learning for game level generation. This is changing it from a classification problem to a reinforcement learning problem. By rewarding the system for generating consistent level or levels with a selected property there is a possibility that new level structures will be generated. There might be no need for such a system to use the original dataset as input.

8 Bibliography

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, http://www.deeplearningbook.org, 2016.

[2] Steve Dahlskog, Julian Togelius, Mark J. Nelson, *Linear levels through n-grams*, 2014.

[3] Julian Togelius, Mark J. Nelson, Alex J, Champandard *Procedural Content Generation: Goals, Challenges and Actionable Steps*, 2013.

[4] Shaker, Noor and Togelius, Julian and Nelson, Mark J. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, 2016.

[5] Horn Britton, Steve Dahlskog, Noor Shaker, Gillian Smith, Julian Togelius *A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework*, 2014.

[6] Bengio, et al. *Learning Long-Term Dependencies with Gradient Descent is Difficult*, 1994.

[7] N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius, and M. O'Neill. *Evolving levels for Super Mario Bros using grammatical evolution* 2012

9 Appendix A: Metrics

Original levels

	Leniency			Linearity			Density			Column Diversity	
Min	Average	Max	Min	Average	Max	Min	Average	Max	Min	Average	Max
-66	-22.96	-4.5	0.43	0.58	0.76	0.60	0.98	1,2	19	33.53	49

9.1 Column based generation

The diversity parameter set to increasing refers to that the diversity is initialized low and is increased when a column equal to the previous is generated.

9.1.1 Subsequence length 3

Diversity	Epoch	Consistency		Leniency			Linearity	
			Min	Average	Max	Min	Average	Max
0,2	100	95	-13.0	-1.205	-0.5	0.86	0.94	0.98
0,2	300	95	-7.5	-1.28	-0.5	0.67	0.93	0.97
0,2	600	99	-11.0	-1.36	-0.5	0.70	0.94	0.98
0,5	100	97	-7.5	-2.4	-0.5	0.69	0.93	0.96
0,5	300	92	-12.5	-3.76	0.5	0.74	0.90	0.96
0,5	600	97	-22.0	-3.44	-0.5	0.79	0.92	0.96
1	100	99	-42.0	-23.885	-14.0	0.37	0.51	0.73
1	300	95	-43.0	-25.965	-14.0	0.37	0.49	0.68
1	600	90	-37.5	-25.06	-12.5	0.44	0.56	0.67
1,2	100	97	-49.0	-33.055	-20.5	0.27	0.36	0.45
1,2	300	96	-52.5	-30.575	-18.0	0.31	0.40	0.49
1,2	600	97	-45.5	-29.625	-18.5	0.35	0.44	0.58
Increasing	100	84	-27.0	-12.2	-3.5	0.63	0.75	0.90
Increasing	300	64	-28.5	-16.335	-6.0	0.56	0.73	0.90
Increasing	600	80	-25.5	-16.575	-5.5	0.57	0.72	0.95

Diversity	Epoch		Density			Pattern Density	
		Min	Average	Max	Min	Average	Max
0,2	100	0.87	1.04	2.99	0	181.76	200
0,2	300	0.74	1.03	2.00	0	177.09	200
0,2	600	0.87	1.01	1.99	0	187.6	200
0,5	100	0.90	1.01	1.98	0	190.26	200
0,5	300	0.86	0.99	1.16	114	191.09	200
0,5	600	0.82	0.99	1.08	72	194.23	200
1	100	0.78	1.00	1.21	39	105.16	152
1	300	0.77	1.00	1.21	30	101.98	166
1	600	0.79	1.01	1.26	68	131.21	179
1,2	100	0.82	1.01	1.22	13	43.6	100
1,2	300	0.76	1.01	1.27	15	61.89	114
1,2	600	0.77	1.02	1.30	48	93.16	139
Increasing	100	0.83	1.00	1.56	15	163.02	194
Increasing	300	0.83	1.00	1.23	106	173.72	200
Increasing	600	0.76	0.99	1.17	132	172.58	193

Diversity	Epoch		Pattern Variation			Column Diversity	
		Min	Average	Max	Min	Average	Max
0,2	100	1	19.24	201	2	3.68	11
0,2	300	1	23.91	201	2	4.22	14
0,2	600	1	13.4	201	2	3.82	17
0,5	100	1	10.74	201	2	4.87	13
0,5	300	1	9.91	87	2	7.36	26
0,5	600	1	6.77	129	2	5.91	25
1	100	49	95.84	162	17	43.49	68
1	300	35	99.02	171	28	46.88	71
1	600	22	69.79	133	24	38.4	57
1,2	100	101	157.4	188	51	76.16	102
1,2	300	87	139.11	186	44	63.82	89
1,2	600	62	107.84	153	27	54.37	74
Increasing	100	7	37.98	186	10	17.96	26
Increasing	300	1	27.28	95	9	19.45	31
Increasing	600	8	28.42	69	6	21.11	34

9.1.2 Subsequence length 10

Diversity	Epoch	Consistency		Leniency			Linearity	
			Min	Average	Max	Min	Average	Max
0,2	100	97	-23.5	-2.215	-0.5	0.42	0.90	0.96
0,2	300	96	-72.5	-11.585	0.5	0.54	0.88	0.96
0,2	600	98	-57.5	-13.2	-0.5	0.50	0.84	0.96
0,5	100	93	-13.0	-3.345	-0.5	0.52	0.87	0.96
0,5	300	88	-67.5	-16.765	-0.5	0.47	0.81	0.96
0,5	600	95	-49.0	-9.47	-0.5	0.50	0.80	0.96
1	100	96	-31.5	-17.31	-4.5	0.37	0.58	0.82
1	300	93	-50.5	-29.4	-16.0	0.37	0.52	0.74
1	600	87	-42.5	-22.495	-10.0	0.39	0.55	0.85
1,2	100	93	-43.0	-28.815	-14.0	0.32	0.42	0.59
1,2	300	88	-52.0	-33.0	-19.0	0.31	0.45	0.60
1,2	600	93	-41.5	-25.935	-14.5	0.35	0.47	0.61
Increasing	100	82	-27.5	-12.105	-4.5	0.34	0.72	0.86
Increasing	300	86	-70.5	-35.875	-8.0	0.40	0.71	0.95
Increasing	600	91	-58.5	-27.12	1.0	0.43	0.62	0.92

Diversity	Epoch		Density			Pattern Density	
		Min	Average	Max	Min	Average	Max
0,2	100	0.82	1.00	1.98	3	196.66	207
0,2	300	0.76	1.02	1.93	8	193.73	207
0,2	600	0.72	0.98	1.46	109	189.65	207
0,5	100	0.72	0.97	1.14	99	191.57	207
0,5	300	0.66	0.98	1.25	99	181.5	207
0,5	600	0.68	0.97	1.16	104	183.15	207
1	100	0.72	0.99	1.16	37	124.41	185
1	300	0.73	1.01	1.25	53	103.79	158
1	600	0.76	0.97	1.20	62	120.37	189
1,2	100	0.78	1.02	1.24	38	70.21	108
1,2	300	0.71	1.03	1.19	23	79.22	126
1,2	600	0.73	1.01	1.29	41	95.41	142
Increasing	100	0.58	0.95	1.09	32	163.65	197
Increasing	300	0.61	0.96	1.14	61	161.18	206

Diversity	Epoch		Pattern Variation			Column Diversity	
		Min	Average	Max	Min	Average	Max
0,2	100	1	11.34	205	2	6.71	31
0,2	300	1	14.27	200	2	9.41	40
0,2	600	1	18.35	99	2	11.61	39
0,5	100	1	16.43	109	2	8.45	33
0,5	300	1	26.5	109	2	14.5	43
0,5	600	1	24.85	104	2	16.22	47
1	100	23	83.59	171	13	31.31	64
1	300	50	104.21	155	27	40.12	68
1	600	19	87.63	146	22	38.36	66
1,2	100	100	137.79	170	30	52.12	77
1,2	300	82	128.78	185	32	51.87	73
1,2	600	66	112.59	167	31	46.19	68
Increasing	100	11	44.35	176	8	20.0	37
Increasing	300	2	46.82	147	5	23.01	55
Increasing	600	1	57.47	142	8	30.11	54

9.1.3 Subsequence length 50

Diversity	Epoch	Consistency		Leniency			Linearity	
			Min	Average	Max	Min	Average	Max
0,2	100	67	-20.0	-7.56	-1.5	0.62	0.83	0.94
0,2	300	83	-62.5	-35.85	-12.5	0.46	0.74	0.92
0,2	600	63	-46.0	-28.22	-7.5	0.41	0.58	0.79
0,5	100	69	-32.5	-11.61	-2.0	0.55	0.81	0.94
0,5	300	69	-53.5	-33.66	-8.5	0.40	0.66	0.82
0,5	600	68	-49.5	-28.145	-12.0	0.37	0.59	0.81
1	100	80	-47.5	-31.065	-17.5	0.39	0.56	0.79
1	300	70	-49.0	-30.84	-12.5	0.35	0.51	0.79
1	600	74	-49.0	-30.04	-13.0	0.35	0.52	0.73
1,2	100	83	-56.0	-36.68	-20.0	0.33	0.46	0.58
1,2	300	83	-50.5	-34.915	-15.5	0.29	0.45	0.59
1,2	600	73	-45.5	-32.715	-16.5	0.30	0.48	0.74
Increasing	100	37	-34.5	-19.405	-6.5	0.33	0.64	0.81
Increasing	300	58	-38.5	-24.02	-5.0	0.45	0.62	0.86
Increasing	600	63	-60.5	-33.905	-14.5	0.44	0.57	0.74

Diversity	Epoch		Density			Pattern Density	
		Min	Average	Max	Min	Average	Max
0,2	100	0.69	0.96	1.16	133	222.65	247
0,2	300	0.68	0.99	1.26	85	199.07	247
0,2	600	0.63	0.92	1.20	99	162.62	226
0,5	100	0.75	0.96	1.15	119	213.86	247
0,5	300	0.57	0.97	1.33	39	175.73	242
0,5	600	0.50	0.93	1.20	51	158.09	219
1	100	0.67	0.97	1.18	76	145.0	209
1	300	0.45	0.97	1.24	43	126.53	186
1	600	0.62	0.95	1.30	57	133.92	192
1,2	100	0.74	1.00	1.31	56	102.31	151
1,2	300	0.63	0.97	1.26	9	103.71	166
1,2	600	0.49	0.95	1.30	37	116.27	174
Increasing	100	0.30	0.93	1.69	85	180.63	228
Increasing	300	0.56	0.92	1.19	87	168.71	231
Increasing	600	0.68	0.95	1.24	69	150.84	218

Diversity	Epoch		Pattern Variation			Column Diversity	
		Min	Average	Max	Min	Average	Max
0,2	100	1	25.35	115	6	15.29	38
0,2	300	1	48.93	163	8	21.69	48
0,2	600	22	85.38	149	17	37.71	66
0,5	100	1	34.14	129	6	17.23	44
0,5	300	6	72.27	209	11	27.44	63
0,5	600	29	89.91	197	19	35.76	62
1	100	39	103.0	172	16	37.97	68
1	300	62	121.47	205	24	47.4	81
1	600	56	114.08	191	25	45.37	70
1,2	100	97	145.69	192	34	55.03	81
1,2	300	82	144.29	239	40	60.72	90
1,2	600	74	131.73	211	29	52.86	83
Increasing	100	20	67.37	163	13	25.74	44
Increasing	300	17	79.29	161	12	29.11	49
Increasing	600	30	97.16	179	18	34.98	57

9.2 Character based generation

9.2.1 Subsequence length 52

Diversity	Epoch	Consistency		Leniency			Linearity	
			Min	Average	Max	Min	Average	Max
0.2	100	30	-3.5	-2.3	-1.0	0.91	0.94	0.95
0.2	300	10	-4.0	-1.85	-1.0	0.93	0.94	0.95
0.2	600	60	-4.5	-1.9	-1.0	0.84	0.93	0.95
0.5	100	40	-5.0	-2.7	-1.0	0.91	0.94	0.96
0.5	300	10	-6.5	-3.15	-1.0	0.81	0.89	0.95
0.5	600	60	-11.0	-4.25	-1.0	0.84	0.92	0.95
1	100	20	-27.0	-19.8	-7.5	0.33	0.59	0.74
1	300	10	-61.0	-27.4	-15.0	0.15	0.48	0.60
1	600	70	-75.0	-33.15	-18.0	0.31	0.60	0.76
1,2	100	0	-42.0	-33.15	-23.5	0.05	0.27	0.48
1,2	300	10	-63.5	-34.75	-17.5	0.14	0.32	0.54
1,2	600	10	-69.5	-55.3	-30.0	0.13	0.28	0.40

Diversity	Epoch		Density			Pattern Density	
		Min	Average	Max	Min	Average	Max
0.2	100	0.93	0.97	0.99	196	198.7	200
0.2	300	0.94	0.98	0.99	0	163.3	200
0.2	600	0.97	0.99	0.99	171	196.3	201
0.5	100	0.95	0.98	0.99	191	197.8	200
0.5	300	0.95	1.02	1.13	163	187.8	200
0.5	600	0.96	0.98	1.00	181	196.7	201
1	100	0.84	0.93	1.05	66	102.4	142
1	300	0.52	1.07	1.41	7	75.1	113
1	600	0.10	0.85	1.08	0	109.3	166
1,2	100	0.88	1.39	2.44	0	19.9	77
1,2	300	0.57	1.07	1.44	3	29.3	75
1,2	600	0.13	0.45	1.07	0	24.7	83

Diversity	Epoch		Pattern Variation			Column Diversity	
		Min	Average	Max	Min	Average	Max
0.2	100	3	4.3	7	4	5.8	7
0.2	300	3	39.7	203	4	6.0	10
0.2	600	2	6.7	32	3	6.4	16
0.5	100	3	5.2	12	4	5.9	8
0.5	300	3	15.2	40	4	9.9	18
0.5	600	2	6.3	22	4	6.7	13
1	100	61	99.7	137	28	39.9	52
1	300	90	147.5	356	36	57.2	119
1	600	37	182.4	1093	21	45.2	98
1,2	100	78	134.9	198	57	78.9	109
1,2	300	128	191.5	328	56	89.8	133
1,2	600	112	474.5	1083	66	99.9	131

9.2.2 Subsequence length 104

Diversity	Epoch	Consistency		Leniency			Linearity	
			Min	Average	Max	Min	Average	Max
0.2	100	40	-13.0	-2.9	-1.0	0.33	0.81	0.95
0.2	300	40	-53.0	-8.65	-1.0	0.88	0.93	0.96
0.2	600	20	-31.0	-5.45	-1.0	0.27	0.85	0.96
0.5	100	30	-13.5	-4.15	0.0	0.42	0.80	0.95
0.5	300	30	-11.0	-3.35	-1.0	0.78	0.92	0.95
0.5	600	10	-28.5	-7.95	-1.0	0.27	0.81	0.96
1	100	20	-39.5	-30.5	-21.0	0.15	0.29	0.35
1	300	50	-32.5	-20.4	-12.0	0.55	0.62	0.70
1	600	50	-33.5	-22.25	-13.0	0.42	0.55	0.75
1,2	100	0	-63.5	-49.9	-39.0	0.18	0.25	0.32
1,2	300	30	-47.5	-31.75	-16.5	0.37	0.47	0.57
1,2	600	30	-31.5	-24.95	-17.5	0.41	0.48	0.58

Diversity	Epoch		Density			Pattern Density	
		Min	Average	Max	Min	Average	Max
0.2	100	0.03	0.80	0.99	73	183.8	204
0.2	300	0.50	0.90	1.01	18	184.0	205
0.2	600	0.02	0.84	1.02	15	179.0	204
0.5	100	0.33	0.87	0.99	22	148.8	204
0.5	300	0.82	0.96	1.01	107	190.4	205
0.5	600	0.02	0.79	1.01	13	151.3	204
1	100	1.03	1.15	1.30	0	6.2	14
1	300	0.78	0.93	1.10	61	118.6	163
1	600	0.34	0.68	0.87	27	75.2	156
1,2	100	1.30	1.40	1.63	0	3.6	11
1,2	300	0.78	0.97	1.10	35	57.9	97
1,2	600	0.66	0.85	0.99	27	56.5	89

Diversity	Epoch		Pattern Variation			Column Diversity	
		Min	Average	Max	Min	Average	Max
0.2	100	3	23.2	134	4	6.2	11
0.2	300	2	22.9	188	2	5.6	9
0.2	600	3	27.8	192	4	6.1	13
0.5	100	3	51.1	179	4	10.5	26
0.5	300	2	16.6	100	3	7.0	16
0.5	600	3	55.7	195	5	9.4	21
1	100	175	196.0	206	99	118.5	156
1	300	44	88.0	145	28	39.9	59
1	600	52	131.9	180	26	47.6	65
1,2	100	189	200.2	215	151	162.0	173
1,2	300	111	148.7	170	56	79.6	103
1,2	600	117	151.0	183	51	64.1	75