Hybrid Weather Simulation System for Real-Time Rendering

Kinga Aszyk*
Supervised by: Marek Wernikowski†

Faculty of Computer Science and Information Technology
West Pomeranian University of Technology
Szczecin / Poland

Abstract

Integrating dynamic weather systems into realistic games is a key step toward enhancing both immersion and gameplay mechanics. However, typical implementations often lack the depth of the real world, relying instead on simple, random transitions between fixed conditions. This paper proposes a novel hybrid generation system, implemented in the Unity engine, designed to better reflect actual weather conditions. The architecture integrates Perlin noise for wind direction, a neural network to determine atmospheric pressure, temperature and wind speed, and Markov chains for weather state selection. To create this model, we introduce a rule-based approach for determining categorical weather states directly from continuous meteorological data. We conducted user testing to evaluate preferences between this realistic model and traditional random systems. The results indicate that increased realism is important, however its impact on the player experience remains heavily dependent on the specific game genre.

Keywords: computer game, weather, simulation, model

1 Introduction

Weather can be defined as the condition of the atmosphere at a particular place and time. Its prediction plays a major role in the everyday lives of people around the world. It includes multiple different factors such as temperature, wind, precipitation, and solar exposure. While its accuracy varies depending on the location and timeframe, it usually gives a good estimate of what one should prepare for in the coming hours or days. Forecasts are usually based on weather models, which typically take into account complex formulas, estimations, and neural networks based on previously acquired data.

While the weather in real world is a complex system that relies on multiple variables, typical weather systems in games lack this depth, being generally limited to a set of fixed conditions that change in a random fashion. Developers usually represent weather by selecting an appropriate skybox, occasionally adding visual effects such as fog, or introducing basic mechanics related to temperature influence. We argue that such solutions lack essential aspects, as the proper implementation of weather systems plays an important role for a variety of reasons. Firstly, from a game design perspective, weather significantly contributes to the player's sense of *immersion* – a psychological state in which the player becomes deeply absorbed in the virtual world. Secondly, weather systems can be used as a meaningful component of *green games*. These types of games aim to raise awareness of climate change and environmental issues, encouraging players to develop a greater concern for nature and sustainability.

In this paper, we propose a novel system that takes into account multiple methods typically used for weather forecasting. Our system takes current conditions as input and extrapolates them based on predictions to continuously change the weather, simulating changes in the virtual environment. As different aspects of the weather require different data and methods, our approach is a hybrid of multiple solutions integrated into a single framework. For most parameters, we use a neural network model, specifically to determine atmospheric pressure, temperature, and wind speed. For wind direction, we utilise an empirically calibrated Perlin noise. To switch between different weather states, we use Markov chains. For the purpose of this work, the term *weather state* is used to describe one of eight predefined atmospheric conditions: sunny, foggy, cloudy, rainy, snowy, stormy, hail, or tornado.

Although the realistic simulation of weather systems is the main goal of this paper, we also identify potential problems with its implementation in video games environments. While the audiovisual and gameplay aspects of the simulation could greatly increase immersion in the virtual world, highly accurate predictions might also become too stale. In general, rapid changes in weather conditions do not occur, which some players might consider too boring for fast-paced games. As such, to test the usability of our system, we also perform a series of tests comparing different weather models. The results are validated through an experimental procedure.

The paper is structured as follows: in Section 2, we provide a theoretical background on meteorological parameters and weather forecasting. In Section 3, we present

*aszyk_kinga@zut.edu.pl
†marek.wernikowski@zut.edu.pl

the methodology of our hybrid simulation framework, describing in detail the data processing and the integration of the GRU network, Perlin noise, and Markov chains. Section 4 provides information on the implementation of the proposed model within the Unity engine, alongside the perceptual study evaluating its quality. Finally, in Section 5, we summarize our findings and outline future work.

Our main contributions include:

- The development of a novel, hybrid weather simulation system, integrating Perlin noise for wind direction, a Gated Recurrent Unit (GRU) neural network for continuous atmospheric parameters, and Markov chains for discrete weather state selection.
- A heuristic methodology to reconstruct the sequence of discrete weather conditions from a dataset of continuous meteorological parameters.
- The implementation of the proposed model within an interactive prototype in the Unity engine.
- A perceptual user study comparing our realistic, Markov-chain-based approach against a purely randomized system.

2 Background and Previous Work

In this section, we discuss the types of weather parameters, previously established methods of predicting weather, and how such methods can be used for simulation.

2.1 Weather Conditions and Parameters

Weather is a complex, dynamic system driven by numerous meteorological parameters. While minor changes often remain unnoticed, major or more extreme conditions can significantly affect human perception and behaviour. In the context of video games, accurately simulating these factors could improve visual fidelity, enhance gameplay mechanics, and greatly impact player immersion.

- **Air pressure:** Commonly associated with human well-being. Significant deviations from the standard sea-level pressure of 1013.25 hPa can trigger physical responses: low pressure can cause headaches, while high pressure can lead to fatigue.
- **Wind speed and wind direction:** Primarily impact the environment, e.g., by causing the rustling of leaves or creating waves on bodies of water. Strong winds can also affect a person's ease of movement, acting as an opposing force.
- **Temperature:** Large changes in temperature can invoke several effects on the human body. Excessively low temperature can cause frostbite, while extreme heat can lead to dehydration or heatstrokes.

Introducing these parameters into a video game could greatly impact the gameplay experience. Simulated headaches could decrease the player's stamina, bad weather could force the player to wear appropriate clothing, and wind could introduce forced movement. Variations in these parameters could also trigger audiovisual effects, like camera shaking, specific audio cues, or vignette effects. Overall, mapping physical weather data to specific in-game mechanics creates a more believable and engaging virtual world.

While raw meteorological data guides the weather system, human perception naturally categorises different conditions into specific states, ranging from clear skies to heavy rain or snow. Furthermore, several atmospheric events, such as hail or a tornado, are more distinct and have a larger impact on the game. In virtual environments, categorising weather into these discrete states plays a major role in player immersion. As with continuous parameters, it allows developers to add specific visual effects, adjust lighting models, and introduce dedicated gameplay mechanics – arguably on a much more profound scale.

2.2 Weather Forecasting

The prediction of weather, until recently, has relied heavily on observational data and complex physical modelling. The primary approach often used is Numerical Weather Prediction (NWP). Models based on NWP simulate weather conditions using non-linear differential equations, typically from the fields of fluid dynamics and thermodynamics. A common solution involves the Navier-Stokes equations [8, 1], which describe changes in atmospheric wind, pressure, density, and temperature. Additionally, General Circulation Models (GCMs) were developed, which focus more on long-term atmospheric behaviour and general patterns on a planet scale [16, 10]. These models are capable of modelling continuous interactions between the atmosphere, oceans, and land surfaces over an extended period.

Although such systems have greatly improved over time, proving to be highly accurate in their prediction and modelling of atmospheric conditions, they remain very computationally expensive. They typically require powerful computers to process weather data, making them entirely unsuitable for real-time applications. Video games are already heavily constrained by graphics quality and gameplay complexity. Introducing another system that would drain resources on a greater scale than complex visual effects would likely have an adverse effect on immersion and overall gameplay satisfaction.

In recent years, the focus has shifted towards data-driven design, where weather forecasting is based not on physical formulas but on previously collected data. Such a shift became possible due to advances in machine learning [4]. With rapid advancements in artificial intelligence, multiple solutions have been proposed, utilising different architectures, including Convolutional Neural Networks [17] (CNNs), Graph Neural Networks [11] (GNNs), Recurrent Neural Networks [9] (RNNs), and Transformers [2]. The cited papers have shown that these models can achieve comparable or better performance than traditional numerical models, while requiring only a fraction

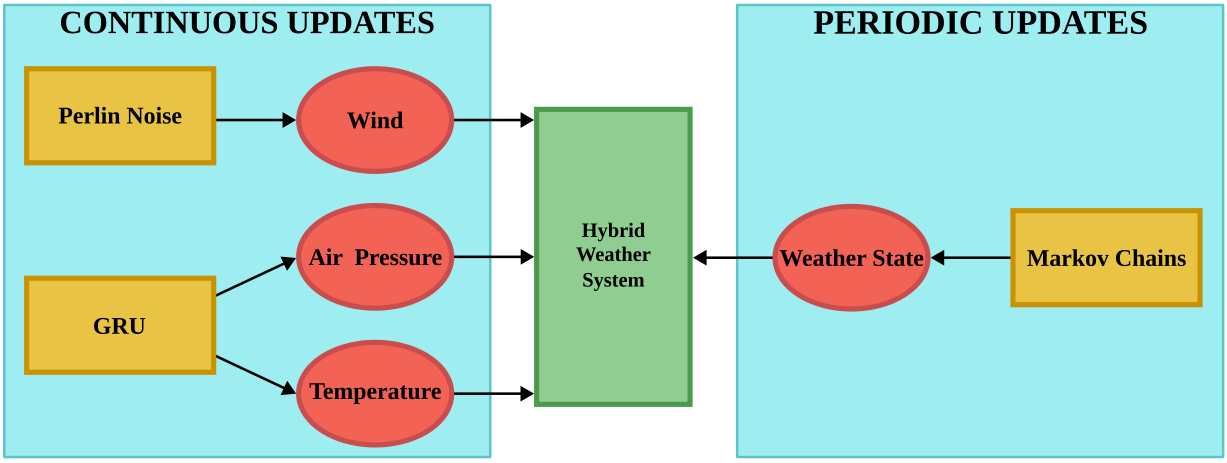

Figure 1: Schematic of the hybrid weather system. The simulated weather parameters depend on data updated each frame (left), and data updated at specific, custom intervals (right).

of the time for inference. Instead of relying on physical processes, neural networks can learn complex, non-linear atmospheric relationships directly from massive amounts of data. In our work, we use a GRU network, which is a type of RNN [3]. It utilises update and reset gates to control the amount of retained and removed information. This architecture allows us to model weather sequences while avoiding the higher computational complexity of standard Long Short-Term Memory (LSTM) networks [5].

Apart from physics models and deep learning, statistical methods have also been used to predict weather patterns and model uncertainty. Markov chains have been utilised to model the probabilistic transitions between discrete weather states, e.g. a rainy and a sunny day [6]. Such a system relies only on the current state to predict the next one, making it a very lightweight solution, suitable for real-time simulation. In this work, we create a Markov chain state system based on the publicly available data.

Other stochastic systems have also been analysed, e.g., by generating noise to represent chaotic weather elements like wind direction or precipitation [15]. An important aspect to consider is the nature of the randomness itself – relying on fully random transitions would create jittery changes in the wind vector, which is unnatural and immersion-breaking for players. As such, we employ Perlin noise, a gradient noise function that yields pseudorandom, smoothly interpolated values. Its continuous, wave-like output mimics the wind in a much more natural way.

2.3 Video Games Weather Systems

Virtual weather significantly impacts player immersion and gameplay mechanics [14]. However, most weather systems in video games prioritise graphical fidelity over accurate meteorological simulation [13]. Typical game engines utilise finite state machines and randomised algorithms, such as Perlin noise, primarily to trigger transitions between visual weather states. These stochastic processes simulate weather changes without relying on underlying meteorological model or actual forecasting. For instance, while titles like *The Witcher 3* and *Red Dead Redemption 2* generate diverse weather conditions, these transitions are driven by scripted probabilities rather than physical atmospheric mechanics.

An alternative approach bypasses in-game generation by fetching real-time meteorological data [7]. This method is utilised in *Microsoft Flight Simulator*, where live data drives the simulation of atmospheric variables such as air pressure, temperature, and wind. While this guarantees realistic condition sequences, it restricts the ability to precedurally adjust the rate or severity of weather changes for gameplay uses. As such, there remains a gap in the development of logic-driven systems capable of forecasting weather natively within the engine, which could lead to more dynamic and immersive environments.

3 Proposed Method

The proposed method combines multiple elements of weather forecasting to create a robust simulation framework. Its architecture is presented in Fig. 1. Wind speed, air pressure, and temperature are computed in real-time based on the output of the GRU network. Due to data limitations, wind direction is randomized with Perlin noise. General weather states (e.g., sunny or rainy) are updated by a Markov chains-based state machine, which is responsible for transitions between conditions according to predefined probabilities. These updates occur at a lower frequency to prevent sudden weather shifts. In this section, all components of this system are described in detail.

3.1 Dataset

The proposed model is based on a dataset obtained from the Meteostat platform [12] for the city of Szczecin, Poland. The available data spans a 50-year period and is

from \ to	Sunny	Cloudy	Rain	Snow	Hail	Storm	Tornado	Fog
Sunny	0.70683	0.09971	0.18306	0.00022	0.00022	0.00979	0.00000	0.00022
Cloudy	0.06736	0.73692	0.14858	0.01634	0.00012	0.01234	0.00000	0.01834
Rain	0.15751	0.19216	0.36950	0.01579	0.00023	0.03852	0.00000	0.22635
Snow	0.00265	0.35450	0.08995	0.27249	0.00265	0.02116	0.00000	0.25661
Hail	0.14286	0.14286	0.14286	0.14286	0.14286	0.14286	0.00000	0.14286
Storm	0.11400	0.29600	0.12600	0.01000	0.00200	0.23200	0.00200	0.21800
Tornado	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000
Fog	0.00120	0.26451	0.23049	0.02401	0.00040	0.02241	0.00000	0.45698

Table 1: Markov chains transition matrix demonstrating the probabilities of changing from one weather state to another.

distributed in CSV format with the following features for every day: average temperature, minimum temperature, maximum temperature, precipitation, snow level, wind direction, wind speed, wind peak gust, air pressure, and total sunshine time. Unfortunately, the dataset exhibits some sparsity, particularly in the earlier records, where some observations are missing.

We used the average temperature where possible. Where it was unavailable, we calculated it using the average of the min. and max. temperatures. If temperature data was entirely missing, the row was omitted. We also utilised precipitation and wind speed data. In both cases, missing values were assumed to be 0.0. For air pressure, missing values were marked as "None". As we use information on pressure changes to predict storms (see Section 3.4), this approach avoids a situation where such a prediction would be reduced to mere guesswork. Although our dataset included a "wind direction" column, it contained only very recent data, which was insufficient for our solution. We did not use data regarding snow level, sunshine duration, or wind peak gust, as they were not part of our proposed weather model.

3.2 Weather Parameter Prediction

The core of the method used to predict weather relies on the GRU network, specifically trained to forecast atmospheric parameters. The main advantage of GRU over standard RNNs is its gating mechanism, which regulates the amount of information passed through the network.

At the t-th step, the network receives as input the hidden state from the previous step (H_{t-1}) and the previously computed parameters: temperature (T_{t-1}), air pressure (P_{t-1}), and wind speed (V_{t-1}). The update gate determines how much of the previous weather state information should be kept, while the reset gate decides how to combine the new input with the previous memory. The output consists of the new weather parameters (T_t, P_t, and V_t) and the hidden state H_t, which serves as the input for the next iteration.

The training of the GRU network required specific preprocessing. To account for season weather changes, the entries from the CSV file were first aggregated into monthly data points. This aggregation was performed to simplify the training process and to capture weather variations resulting from seasonal changes. Months with no data were removed, while any remaining missing values were filled using the median of the respective feature. For stable convergence, the input data was scaled to a $[0,1]$ range. We employed a sliding window technique with a look-back period of 10 data points. We used the Adam optimizer for model optimisation and Mean Squared Error as a loss function. In total, we used 20,752 rows of daily data, aggregated into 886 monthly rows. We concluded the training after 700 epochs.

In the conducted experiments, a custom model implementation was developed in Python using the PyTorch, NumPy, Pandas, and scikit-learn libraries. The model was trained with a learning rate of 1×10^{-3}, 64 hidden units, and a dropout rate of 0.1. In addition, weight decay regularisation of 1×10^{-4} and gradient clipping with a threshold of 1.0 were applied. The input data were represented as sequences of length 10, meaning that each sample had the shape (batch, 10, F), where F denotes the number of input features; in the analysed case, the input size was equal to 8. The prediction horizon was set to 12. The training process was performed on 80% of the prepared time windows, while the remaining 20% were used for validation. No separate test set was included in the experimental setup. To adaptively adjust the learning rate during training, the ReduceLROnPlateau scheduler was employed, whereas early stopping was not used because the patience parameter was set to None.

3.3 Wind Direction

Due to data limitations and the inherent unpredictability of wind direction, we opted for a more pragmatic approach. Instead of computing it directly from spatial pressure gradients, our model utilises a Perlin noise function. This method, while being only an approximation, is highly computationally efficient and yields physically and temporally plausible results.

We sampled a 1D slice of 2D Perlin noise over time, ensuring smooth and continuous transitions. The sampling process is defined by the following formula:

$$n(t) = \text{Perlin}(t \cdot s, y_0),\ n(t) \in [0,1] \tag{1}$$

Figure 2: Examples of visual representations of different states caused by suboptimal weather conditions: headache, feeling cold, and fatigue.

where:

- t – current time step,
- s – scale governing the frequency of changes,
- y_0 – a fixed coordinate, acting as the noise seed.

To achieve physically plausible results, the parameters were empirically tuned. The temporal scale was set to $s = 1000$, and the fixed seed was chosen as $y_0 = 1.1$. Finally, the normalized continuous output $n(t)$ was mapped to a full angular range (from 0 to 360 degrees) to calculate the actual wind direction.

3.4 Discrete Weather states

Although the continuous simulation of weather parameters provides the basis of the simulation, discrete weather states have the most immediate impact on visual aesthetics and gameplay mechanics. In this paper, the transitions between these discrete conditions are modelled using Markov chains. Each weather condition is treated as an individual state. In every iteration, the system either remains in its current state or transitions to another based on predefined probabilities. To facilitate this, transition probabilities are defined for every possible pair of basic weather states, forming a transition matrix.

As mentioned in Section 3.1, the available historical dataset is limited to continuous meteorological parameters and lacks weather state labels. To construct the Markov chain transition matrix, it was first necessary to reconstruct the historical sequence of discrete weather conditions. We inferred these states by applying a set of deterministic heuristic rules to the continuous data. By default, the weather for any given time step is classified as sunny. This is subsequently changed to another state according to the following criteria:

- **Storm:** occurs when there is a positive day-to-day pressure change, defined as greater than 6.0 hPa.
- **Tornado:** requires a preceding storm state. Consecutive tornado states are prohibited.
- **Snowy:** occurs only with recorded precipitation and if the temperature is below 0 °C.
- **Foggy:** appears during cold temperatures if it has rained within the last 3 days. We defined cold temperatures as between -2.0 °C and 8.0 °C.
- **Hail:** occurs when precipitation exceeds 1.0 mm/day and wind speed exceeds 12.0 m/s.
- **Rainy:** occurs with recorded precipitation and when the temperature is above 0 °C.
- **Cloudy:** first, the model checks more specific weather states (hail, storm, rainy/snowy, and foggy). If none of their conditions are met, it uses a simplified rule to choose between sunny and cloudy: when the temperature is at least 15 °C, the state is sunny; otherwise, it is cloudy.

By applying these rules, which are our simplified interpretation of real-word rules, we assigned a discrete weather state for every entry in the database. By calculating the relative frequencies of all transitions between states, we computed the transition matrix. The results of our calculations are presented in Tab. 1.

4 Implementation and Evaluation

To validate the proposed weather model and demonstrate its practical applicability, we created an interactive prototype. We designed a scene featuring a lake, a forest, and animated objects. In this section, we describe the implementation details, focusing on the integration of the model and the generation of corresponding visual effects. Furthermore, we present the methodology and results of a user study that we conducted to evaluate perceptual preferences regarding different aspects of the weather simulation.

4.1 Audiovisual Implementation

The model has been integrated into a 3D environment created with the Unity engine. We chose the High Definition Render Pipeline (HDRP) as the rendering backbone to ensure the highest visual fidelity. Wind direction is calculated in each frame using the Perlin noise function. Other weather parameters are estimated in real-time by the GRU network which runs continuously throughout the game.

Changes in parameter values and weather states are reflected in the game as well. While our application serves

Figure 3: Sample images of every weather state shown in the prototype environment.

merely as an example, it provides a solid baseline for future projects. We added a vignette effect to symbolize fatigue, increased the bloom effect and camera shake to represent a headache, and introduced visible breath clouds during low temperatures. We established the thresholds for triggering these specific effects based on typical human responses, as described in Section 2.1. Some of these visual effects are presented in Fig. 2.

The weather changes after a specified time interval, determined by a set variable, according to the Markov chain matrix. If the realistic model is used, the values are set according to Tab. 1. In the random case, all probability values in the matrix are equal, summing to 1 in each row. All weather states are shown in Fig. 3. Sunny weather is created by including forest ambient sounds and bright lighting. Cloudy weather switches the skybox to a darker one, covered in clouds. Hail, rain and snow introduce visual precipitation effects in the form of hailstones, raindrops, and snowflakes. Foggy weather decreases the lighting level and obscures distant objects. During stormy weather, lightning bolts appear at irregular intervals, randomly setting trees on fire. Tornados feature special sound cues, visual effects, and gameplay mechanics that push the player.

4.2 User Study

A primary objective of our work was to determine whether a realistic, Markov-chain-based approach of weather state transitions is more suitable for video games than a purely randomised approach. To this end, we conducted a perceptual study in which participants could see both systems in action and compare them directly. As this experiment was specifically designed for the purposes of this study, there are no directly comparable studies available in the literature.

The experiment was divided into two phases. In each phase, participants were shown a fully realised 3D prototype environment, in which they could move freely. Throughout the session, the weather transitioned automatically every 30 seconds. After 4 minutes, the session ended.

The logic governing the weather transitions depended on the experiment conditions. One condition utilised the calculated transition matrix (see Section 3.4), while the other employed a uniform random distribution, with equal probabilities for all weather states. The order in which both parts were shown to the participants was randomised. After the experiment, participants completed a questionnaire in which they answered key questions regarding the weather simulation. The form also included the questions listed in the first columns of Tab. 2 and 3.

A total of 20 participants (14 male, 6 female), ranging in age from 9 to 50 years, took part in the experiment. They all reported at least moderate experience with video games. All participants were naïve to the nature and purpose of this experiment. They were also not informed which model was realistic and which was random.

4.3 Results

We evaluated the results using the data gathered from the questionnaire. The first four questions were dichotomous and captured model preferences in the context of video games. We assessed preferences to determine where such specific systems would be most useful.

Detailed results are shown in Tab. 3. The last column shows the p-value of the binomial test where the null hypothesis assumes that both approaches are equally preferred. The collected data reveals an interesting phenomenon: while it is unclear which variant was preferred in general, the realistic model was a clear winner when the question was aimed specifically at video games. This suggests that although complete randomness might be desired

Question	Mean	95% CI
To what extent did the weather system appear natural (i.e., did it behave similarly to real-world weather)? (Realistic variant)	3.65	[3.15, 4.10]
Did you perceive the sequence of weather states as logically coherent (e.g., without completely implausible transitions)? (Realistic variant)	3.65	[3.05, 4.20]
To what extent did the randomness of the weather system feel appropriate?	3.20	[2.55, 3.80]
How would you rate the post-processing effect simulating the impact of atmospheric pressure during gameplay?	3.85	[3.35, 4.30]
How would you rate the forced player movement effect simulating the impact of wind?	3.50	[2.90, 4.05]
How would you rate the camera shake effect simulating the impact of atmospheric pressure?	3.85	[3.25, 4.40]
How would you rate the visual effect simulating the impact of temperature?	3.45	[2.90, 3.95]

Table 2: Questions asked to the participants after the experiment. Each could be answered on a scale from 1 to 5.

Question	Realistic	Random	p-value
Which variant, in general, did you like more?	55%	45%	0.8238
Which variant would be better in RPGs or any other game with fantasy world?	50%	50%	1.0000
Which model would be better for games simulating reality?	**95%**	5%	< 0.0001
Which model do you prefer in games?	**85%**	15%	0.0026

Table 3: Results of the forced-choice questions about the model preferences. Values in bold indicate statistically significant preferences.

due to its unpredictability and visual attractiveness, players prefer more stability in typical gameplay scenarios. While the realistic model is not as "flashy" as the random one, its predictable results likely feel more familiar to the player, making it the preferable option in gaming. The realistic model was also significantly preferred in simulation settings, which confirms that users correctly distinguished between the two options. The question regarding preferences in RPG games yielded no clear winner, suggesting that participants were conflicted between maintaining realism and creating something unexpected that fits fantasy worlds. Further testing is necessary, involving more specific questions regarding different gameplay expectations.

The other set of questions required participants to rate specific aspects of the simulation on a 5-point scale. The results from this study are presented in Tab. 2. To ensure that our findings provide a reliable estimate of the population mean, the final column reports the 95% Confidence Intervals, calculated using a bootstrap resampling method with 100,000 iterations.

In the first two questions, participants were asked to rate how natural and logical the realistic state changes were. While both scores were above average, these evaluations were likely skewed by experimental limitations. Specifically, the limited time available to present the specific weather states might have impacted the final scores. The third question, regarding the appropriateness of randomness in video games, received more mixed results. This suggests that pure randomness in weather state selection is not the optimal solution, and that implementing a set of rules – even if not strictly based on reality – would benefit gameplay. The remaining questions focused on user preferences regarding our custom implementation of various weather effects (see Section 4.1). All effects received above-average scores, indicating a positive reception of our proposed designs by the participants.

5 Conclusions

In this paper, we presented a novel, hybrid weather simulation system designed for real-time applications in video games. By integrating a GRU neural network, Perlin noise, and Markov chains, we successfully developed a framework that generates coherent and plausible weather patterns. Our solution moves beyond the simple, purely randomised approach, which is often used in the industry.

The results of our perceptual user study demonstrated that players can clearly distinguish between realistic and random weather transitions, and judge them based on their specific contexts. While the Markov-chain-based approach was considered more natural, the overall preference for realism proved to be heavily dependent on the specific game genre. Participant feedback indicated that our custom model is highly suitable for simulation, where immersion heavily relies on correspondence to the real world. On the other hand, in fantasy games, players often preferred the unexpected and dynamic transitions provided by the randomised system, focusing more on changing visual stimuli than on meteorological accuracy.

Future Work

Although the proposed hybrid system proved to be viable, several of its aspects could be improved.

First, the continuous parameter forecasting and discrete

state selection could be more tightly integrated. Currently, the Markov chain relies on a pre-calculated transition matrix that depends solely on the previous weather state. In future work, we could move towards a conditional Markov model, where transition probabilities are dynamically weighted by the meteorological parameters generated by the GRU network. By connecting both aspects of the hybrid system, the simulation could generate physically accurate conditions without relying on specific unchangeable rules.

Second, the visual representation of the available weather states could be expanded. That could include introducing multiple sub-states for discrete weather conditions (such as varying intensities of rain) to make the simulation feel more continuous. Additionally, visual quality could be improved by integrating secondary atmospheric phenomena, such as rainbows.

Finally, although the system already operates in real-time, its performance should be profiled and optimised. Current work focused only on developing a proof of concept system, and optimization was not a priority. As such, optimising the graphics pipeline to maximise performance without negatively impacting the established visual quality remains a key focus for future development.

References

[1] P. Bauer, A. Thorpe, and G. Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, Sept. 2015.

[2] K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian. Accurate medium-range global weather forecasting with 3D neural networks. *Nature*, 619(7970):533–538, July 2023.

[3] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[4] P. D. Dueben and P. Bauer. Challenges and design choices for global weather and climate models based on machine learning. *Geoscientific Model Development*, 11(10):3999–4009, Oct. 2018.

[5] F. Furizal, A. Fawait, H. Maghfiroh, A. Ma'arif, A. Firdaus, and I. Suwarno. Long short-term memory vs gated recurrent unit: A literature review on the performance of deep learning methods in temperature time series forecasting. *International Journal of Robotics and Control Systems*, 4(3):1506–1526, 2024.

[6] Gabriel and Neumann. A Markov chain model for daily rainfall occurrence at Tel Aviv. *Quarterly Journal of the Royal Meteorological Society*, 1962.

[7] A. Grönroos. Real-time weather in gaming : an API-driven approach to dynamic weather. Bachelor's thesis, Turku University of Applied Sciences, 2025.

[8] E. Kalnay. *Atmospheric Modeling, Data Assimilation, and Predictability*. Cambridge University Press, New York, 2003.

[9] Z. Karevan and J. A. Suykens. Transductive LSTM for time-series prediction: An application to weather forecasting. *Neural Networks*, 125:1–9, May 2020.

[10] D. Kochkov, J. Yuval, I. Langmore, P. Norgaard, J. Smith, G. Mooers, M. Klöwer, J. Lottes, S. Rasp, P. Düben, S. Hatfield, P. Battaglia, A. Sanchez-Gonzalez, M. Willson, M. P. Brenner, and S. Hoyer. Neural general circulation models for weather and climate. *Nature*, 632(8027):1060–1066, Aug. 2024.

[11] R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, A. Merose, S. Hoyer, G. Holland, O. Vinyals, J. Stott, A. Pritzel, S. Mohamed, and P. Battaglia. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, Dec. 2023.

[12] C. S. Lamprecht. The Weather's Record Keeper | Meteostat, Feb. 2026. [Online; accessed 20. Feb. 2026].

[13] C. Mudlapur and O. P. Singh. A Review of Realistic Weather Simulation Systems in Video Games: From Scripted Skies to Dynamic Storms. *IEEE Transactions on Games*, 18(1):15–29, Dec. 2025.

[14] S. Roberts and D. Patterson. Virtual weather systems: measuring impact within videogame environments. In *ACM Other conferences*, pages 1–7. Association for Computing Machinery, New York, NY, USA, Jan. 2017.

[15] R. B. T. N. Palmer. Stochastic parametrization and model uncertainty. https://www.ecmwf.int/en/elibrary/75936-stochastic-parametrization-and-model-uncertainty, 200910–2009.

[16] W. M. Washington and C. Parkinson. *Introduction to Three-Dimensional Climate Modeling*. University Science Books, June 2005.

[17] J. A. Weyn, D. R. Durran, and R. Caruana. Improving Data-Driven Global Weather Prediction Using Deep Convolutional Neural Networks on a Cubed Sphere. *Journal of Advances in Modeling Earth Systems*, 12(9):e2020MS002109, 2020.