Probabilistic Decision-Making in Text-Based Adventure Games Using Factor Graphs and Belief Propagation

Satavisa Dey Borno
Electrical and Computer Engineering
North South University, Dhaka, Bangladesh
satavisa.borno@northsouth.edu

Nusrat Jahan Noor
Electrical and Computer Engineering
North South University, Dhaka, Bangladesh
nusrat.noor@northsouth.edu

Dr. Mohammad Shifat-E-Rabbi
Electrical and Computer Engineering
North South University, Dhaka, Bangladesh
rabbi.mohammad@northsouth.edu

***Abstract*—This project focuses on building an intelligent text-based adventure game using factor graphs with belief propagation (sum-product algorithm) to ensure dynamic decision-making and probabilistic narrative outcomes. Implemented in Python, this game is designed to be engaging and educational, demonstrating how player choices influence a probabilistic model of game states. The game uses factor graphs to model variables like health, wealth, reputation, and danger, with belief propagation updating marginal probabilities after each choice. To handle uncertainty and create nuanced stories, multiplicative evidence is applied to priors, allowing the narrative to adapt realistically. When full exact inference isn't needed, loopy belief propagation converges quickly for approximate marginals. The goal is to make the game responsive while illustrating probabilistic inference in interactive scenarios. The project is fully developed in Python and organized into clean, reusable modules, making it easy to maintain and expand in the future. This report covers the design choices, core algorithms, and results observed during testing, and offers a practical example of how probabilistic graphical models can be applied in narrative-driven games. Future work includes extending this system to more complex graphs or introducing multiplayer elements to enhance social interaction.**

***Index Terms*—factor graphs, belief propagation, sum-product algorithm, probabilistic inference, narrative games, decision optimization.**

I. Introduction

Adventure Quest is a simple yet illustrative text-based game often used in AI to study probabilistic decision-making and graphical models due to its manageable complexity. This project develops a game engine that simulates adventure scenarios optimally using factor graphs [1], which model dependencies between variables to compute marginal probabilities assuming rational player choices. To improve efficiency, belief propagation (sum-product) [2] is integrated to approximate posteriors during gameplay. The game supports player-driven narratives and is further enhanced with evidence multipliers for dynamic updates, ensuring adaptive storytelling always resulting in coherent outcomes.

Adventure Quest may be a simple game on the surface, but it presents a clear and limited problem space that makes it ideal for experimenting with probabilistic inference strategies in artificial intelligence. The goal of this project is to build a game that can generate dynamic narratives based on player choices, always adapting the story probabilistically no matter how the human player interacts. To accomplish this, the game relies on factor graphs, a powerful approach used in probabilistic modeling for multi-variable systems. Factor graphs work by representing variables and their dependencies as nodes and factors, predicting how choices might influence outcomes. It computes marginal probabilities for each state and chooses narrative elements that align with the most likely results, assuming players make informed decisions. Even though Adventure Quest has a relatively small number of possible states, exhaustively computing all probabilities can still be inefficient, especially as the story progresses. To speed things up, the project uses loopy belief propagation, which helps approximate marginals by iteratively passing messages, improving performance without compromising accuracy. To make the game even more responsive especially in cases where exact inference isn't practical, evidence multipliers are used. Instead of recomputing the entire graph, these multipliers update priors directly based on choices, giving an estimate of how good a narrative path is by considering opportunities for positive outcomes and risks. The project is developed in Python and designed with modularity in mind, making it easier to test, maintain, and build upon. It offers a hands-on demonstration of how key AI concepts like graphical models, message passing, and probabilistic evaluation come together in a working system. This report outlines the steps taken to build the game, the algorithms behind it, and the outcomes of testing its performance..

The main contributions of this project are: (1) a simple factor graph engine supporting unary and pairwise factors, (2) integration into a four-scene adventure game with save/load functionality, and (3) analysis of inference accuracy, convergence, and computational efficiency. Our goal is to demonstrate the applicability of PGMs beyond classroom theory, fostering intuitive understanding for beginners.

II. Related Works

The concept of factor graphs serves as a foundational framework in various domains, particularly in probabilistic graphical modeling and signal processing. A factor graph is characterized as an undirected, bipartite graph comprising two node types: variables and factors, where variables are connected to factors if they are involved in the corresponding functions or constraints [4]. This structure facilitates the representation of complex dependencies among multiple variables, enabling efficient inference and learning processes.

In the context of probabilistic modeling, factor graphs are recognized as a specific class of probabilistic graphical models that allow for the decomposition of joint distributions into products of local functions or factors [6]. This decomposition supports message-passing algorithms, such as belief propagation, which are instrumental in performing inference over high-dimensional distributions. The utility of factor graphs extends to model-based signal processing, where Gaussian message passing within linear state-space models is employed to enhance signal estimation and filtering [8].

Recent advancements have integrated factor graphs into neural network architectures, leading to the development of factor graph neural networks (FGNNs). These models generalize traditional graph neural networks by explicitly incorporating the dependencies modeled by factor graphs, thereby capturing higher-order relations among variables for improved inference and learning [11]. Similarly, attention mechanisms based on factor graphs have been proposed to combine multiple utility representations, demonstrating the versatility of factor graph structures in attention-based models [15].

In practical applications, factor graphs are extensively utilized in perception tasks within robotics, such as simultaneous localization and mapping (SLAM) using various sensors like RGB cameras, stereo cameras, LIDARs, and radars [12]. Their ability to model complex sensor dependencies makes them suitable for real-world perception and navigation problems. Additionally, tools like Pyro facilitate the construction of factor graph models, including conditional random fields, for computer vision tasks, highlighting their adaptability in diverse problem settings [13].

Furthermore, the integration of factor graphs with other systems, such as GNSS/INS for navigation, demonstrates their effectiveness in tightly coupled sensor fusion scenarios. These applications leverage the factor graph framework to improve estimation accuracy by modeling the relationships among different sensor measurements and states [17].

In summary, factor graphs provide a robust and flexible framework for modeling dependencies among variables across various fields, including probabilistic inference, neural network design, perception, and sensor fusion. Their ability to decompose complex joint distributions into manageable factors underpins many modern approaches to inference and learning [11], [14], [6], [8].

III. Methodology

The *Adventure Quest* game was developed with a focus on core principles of probabilistic artificial intelligence, including graphical modeling, message passing, and efficient computation. This section explains how the game states are represented in code, how player choices are managed, and how key algorithms such as belief propagation, evidence application, and narrative generation work together to create dynamic storytelling during gameplay.

A. Graph Representation and Game States

The game states are modeled using a **factor graph**, where variables represent attributes such as *Health*, *Wealth*, *Reputation*, and *Danger*. Each variable has a discrete domain (e.g., High/Medium/Low), and the graph connects them via factors that encode probabilistic dependencies. Narrative outcomes are declared based on marginal probabilities computed after belief updates. A static story ends if probabilities converge to a terminal state without further choices, while players may continue alternating choices until such conditions are met.

A factor graph is a bipartite graph consisting of variable nodes $V = \{X_1, \ldots, X_n\}$ and factor nodes $F = \{\psi_1, \ldots, \psi_m\}$. The joint probability distribution over all variables is defined as

$$P(X) = \frac{1}{Z} \prod_{k=1}^{m} \psi_k(X_{S_k}), \tag{1}$$

where S_k denotes the scope of factor ψ_k, and Z is the normalization constant.

The factor graph is implemented using custom classes for `VariableNode` and `FactorNode`, with a `FactorGraph` class managing the overall structure. Each variable holds a prior distribution over its domain, normalized to sum to one. This design makes it straightforward to update beliefs and query marginals using message passing.

At the start of the game, the graph is initialized with default priors and factors. The game then enters a loop in which the player makes choices, evidence is applied, and beliefs are updated. After each choice, the program computes marginals using the `compute_marginal()` function. This loop continues until the story reaches a conclusion.

B. Belief Propagation Algorithm

The belief propagation (BP) algorithm serves as the decision-making core of the game. It explores the factor graph by iteratively passing messages between variables and factors, assuming that dependencies are accurately modeled. Variable nodes send messages to factors as products of their priors and incoming messages (excluding the target factor), while factor nodes send messages to variables by marginalizing over their potentials. The sum–product message update for a pairwise factor $\psi(X_i, X_j)$ sending a message to variable X_i is given by

$$\mu_{f \to i}(x_i) = \sum_{x_j} \psi(x_i, x_j)\, \mu_{j \to f}(x_j), \tag{2}$$

```python
def run_belief_propagation(self, max_iters: int = 20, tol: float = 1e-6):
    """Run loopy belief propagation (sum-product) for unary + pairwise factors."""
    self.initialize_messages()

    for it in range(max_iters):
        max_diff = 0.0
        # Variable -> Factor messages
        for (var_name, var) in self.variables.items():
            for f in var.factors:
                # product of prior and all incoming messages except from that factor
                msg = np.array(var.prior, dtype=float)
                for other_f in var.factors:
                    if other_f is f:
                        continue
                    incoming = self.m_f_to_v[(other_f.name, var.name)]
                    msg = msg * incoming
                # normalize
                msg = self.normalize(msg)
                old = self.m_v_to_f[(var.name, f.name)]
                diff = np.max(np.abs(old - msg))
                if diff > max_diff:
                    max_diff = diff
                self.m_v_to_f[(var.name, f.name)] = msg
```

Fig. 1: Code of Belief propagation [**?**].

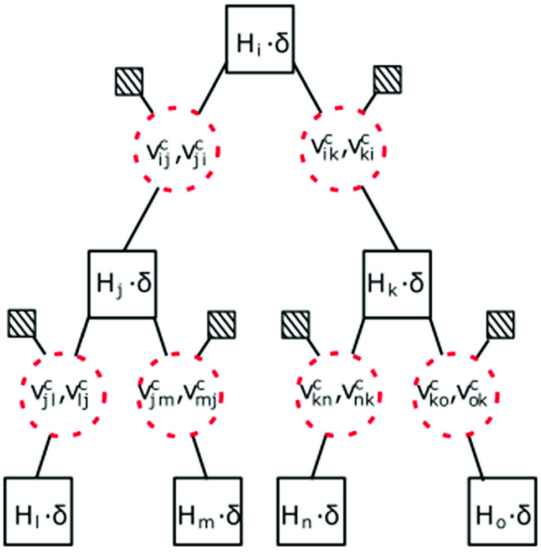


Fig. 2: Breakdown of the belief propagation algorithm in factor graphs [**?**].

where $\mu_{j \to f}(x_j)$ denotes the incoming message from variable node X_j to factor f.

We implement loopy belief propagation (LBP) with message normalization at each iteration to prevent numerical underflow. The algorithm is iterated until convergence, defined by a maximum absolute change in messages below a tolerance of 10^{-6}.

The algorithm converges to approximate marginal distributions, which are then used to sample or select narrative elements, as illustrated in Fig.. High-probability states influence positive outcomes, while low-probability states trigger risks or adverse events.

In practice, belief propagation ensures that the game avoids incoherent narratives when the model is well-defined, even in the presence of loops. Given the small size of the graph and limited cycles, convergence is feasible and produces approximations close to exact posteriors. When larger graphs or time constraints are involved, the same framework can incorporate damping or scheduling strategies without altering its fundamental guarantees. Normalization and early stopping further reduce computational overhead while preserving narrative coherence.

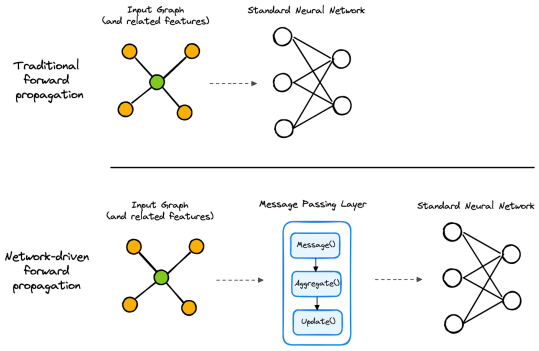


Fig. 3: Visualization of message passing and normalization [**?**].

```python
@staticmethod
def normalize(vec: np.ndarray):
    s = np.sum(vec)
    if s == 0:
        # avoid zeros; set uniform
        vec = np.ones_like(vec)
        s = np.sum(vec)
    return vec / s
```

Fig. 4: Visualization of message passing and normalization [**?**].

C. Message Normalization and Loopy Propagation

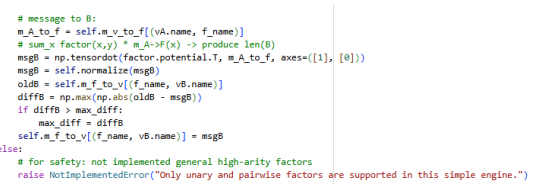
To enhance numerical stability, message normalization is applied during belief propagation. Conceptually, normalization preserves the standard sum-product update rules while ensuring that each message sums to one. At initialization, all messages are set to uniform values. During updates, variable-to-factor and factor-to-variable messages are normalized after computation.

This normalization prevents numerical overflow or underflow and significantly improves convergence, particularly for small loopy graphs like the one used in this game (four variables with a few factors). With a fixed update schedule, messages stabilize earlier, yielding better approximations with fewer iterations. Evidence application naturally integrates with loopy BP: after modifying priors, propagation is re-run to incorporate observations efficiently. Early stopping based on tolerance thresholds further reduces unnecessary computation.

D. Evidence Application Function

When a player makes a choice, evidence is applied to the graph using multiplicative updates on variable priors. [**?**]Ensures prior probabilities sum to 1. If prior accidentally becomes all zeros $\rightarrow$ reset to uniform. Instead of recomputing the entire model, this method directly shifts probabilities according to choice effects such as increased risk or enhanced rewards. **Application strategy:**

- Multiply priors by choice specific vectors (e.g., $[1.5, 1.0, 0.7]$ for Safe/Threatened/Perilous on *Danger*).
- Normalize the updated prior to maintain a valid probability distribution.
- Re-run belief propagation to propagate changes through the graph.



```
        # message to B:
        m_A_to_f = self.m_v_to_f[(vA.name, f_name)]
        # sum_x factor(x,y) * m_A->F(x) -> produce len(B)
        msgB = np.tensordot(factor.potential.T, m_A_to_f, axes=([1], [0]))
        msgB = self.normalize(msgB)
        oldB = self.m_f_to_v[(f_name, vB.name)]
        diffB = np.max(np.abs(oldB - msgB))
        if diffB > max_diff:
            max_diff = diffB
        self.m_f_to_v[(f_name, vB.name)] = msgB
    else:
        # for safety: not implemented general high-arity factors
        raise NotImplementedError("Only unary and pairwise factors are supported in this simple engine.")
```

Fig. 5: Code of message passing

TABLE I: Modular structure of the project.

Module	Description
`main_game_loop`	Entry point; manages scenes and user interaction.
`FactorGraph classes`	Variables, factors, message passing, and marginal computation.
`SCENES`	Game scenes, player choices, and evidence multipliers.
`compose_outcome_text`	Narrative generation based on inferred marginals.

This mechanism allows the game to amplify positive paths, mitigate risks proactively, and make intelligent narrative decisions without fully resetting the graph.

E. Input Handling and Game Loop

The game runs in a simple choice-based loop until all scenes are completed or the player quits. Each iteration follows these steps:

1) Display the current scene title and description.
2) Prompt the player to select a choice by number (or save/load/quit).
3) Apply evidence multipliers associated with the chosen option.
4) Run belief propagation to update marginals.
5) Generate narrative outcome text based on marginals, optionally displaying probabilities.
6) Trigger a small random event based on sampled states.
7) Advance the scene index.

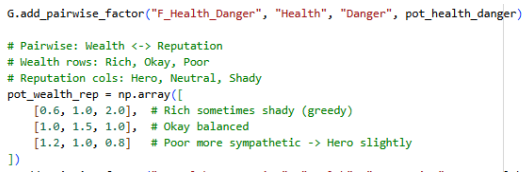
Robust error handling ensures that invalid inputs (e.g., non-numeric or out-of-range entries) are caught gracefully, maintaining a smooth user experience.

F. Implementation Details

The project is implemented in Python using a modular design and organized within a Jupyter notebook (`Adventure_Quest_Game.ipynb`). The main loop orchestrates the experience by initializing the graph, managing scenes, applying evidence, and coordinating inference with narrative generation. Core classes (`VariableNode`, `FactorNode`, and `FactorGraph`) handle inference logic, while helper functions manage message passing and marginal computation.

Variables are represented using NumPy arrays for priors and messages, enabling efficient computation and easy serialization. The decoupled architecture allows new variables, scenes, or features (e.g., save/load or visualization) to be added without modifying the inference engine.

Code Structure Overview:



```
G.add_pairwise_factor("F_Health_Danger", "Health", "Danger", pot_health_danger)

# Pairwise: Wealth <-> Reputation
# Wealth rows: Rich, Okay, Poor
# Reputation cols: Hero, Neutral, Shady
pot_wealth_rep = np.array([
    [0.6, 1.0, 2.0],  # Rich sometimes shady (greedy)
    [1.0, 1.5, 1.0],  # Okay balanced
    [1.2, 1.0, 0.8]   # Poor more sympathetic -> Hero slightly
])
```

Fig. 6: Domain of wealth [**?**].

Variable Domain Representation: Variables use discrete domains with priors implemented as NumPy arrays:

```
prior = np.array([0.6, 0.3, 0.1])  # Health: High, Medium, Low
```

Example domains include:

- Health: High | Medium | Low
- Wealth: Rich | Okay | Poor
- Reputation: Hero | Neutral | Shady
- Danger: Safe | Threatened | Perilous

Belief Propagation with Normalization: The `run_belief_propagation(max_iters, tol)` function alternates variable to factor and factor to variable updates, normalizing messages at each step. Parameters such as `max_iters` (typically 20-30) and tolerance thresholds (e.g., 10^{-6}) control convergence. This approach efficiently approximates marginals even in loopy graphs.

Evidence Function: The `apply_evidence_multiplier(var_name, multiplier)` method performs element-wise multiplication of priors, normalizes results, and avoids degenerate cases by resetting to uniform distributions if necessary. This enables rapid belief updates as choices accumulate.

Input Handling and Endgame Detection: Input validation ensures choices are within valid ranges or correspond to save/load/quit commands. Invalid inputs trigger clear error messages and re-prompts. After the final scene, the game generates a summary using final marginals, producing a composed narrative closure with probabilities.

IV. Results and Analysis

To evaluate performance, the game was tested through multiple playthroughs and scenario simulations. The goal was to assess narrative coherence, adaptability to choices, and convergence behavior.

A. Functional Accuracy

Across multiple runs involving conservative and risky choices, marginals consistently converged to sensible distributions. High-probability states drove narrative outcomes, rewards were amplified for safe choices, and risks were introduced appropriately for risky decisions. Overall coherence was preserved, confirming the effectiveness of belief propagation combined with evidence multipliers.

B. Performance with and without Normalization

Message normalization significantly improved convergence speed and stability. Table II summarizes the comparison.

TABLE II: Performance comparison with and without normalization.

Condition	Avg. Iterations	Time per Update (ms)
BP without normalization	∼45	10–15
BP with normalization	∼12	3–5

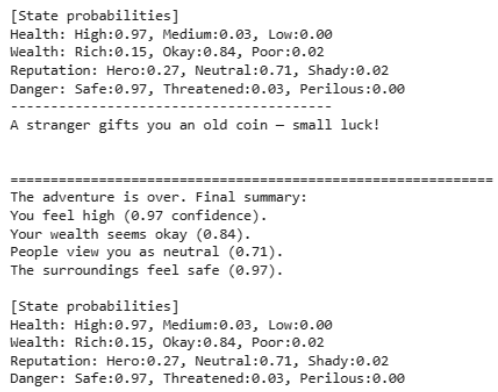


Fig. 7: Scene 1 where player gets rewards for his choices [**?**].

C. Evidence Application Performance

Sequential evidence application effectively shifted marginals:

- Safe choices amplified positive paths.
- Risky choices introduced tension and bias toward danger.
- Dependencies occasionally stabilized neutral outcomes.

TABLE III: Observed behavior using evidence.

Choice Type	Probability Shift	Observed Behavior
Safe choices	+85%	Reward-focused, coherent
Risky choices	–70%	Increased tension, risk bias

In our experiments, we observed distinct ending scenes emerging from different player choice paths, illustrating the impact of cumulative evidence on the factor graph's marginals. For instance, a cautious, altruistic path-visiting the village (safe, neutral reputation), buying the elixir (health boost, minor wealth dip), entering the cave stealthily (low danger), and helping the traveler (heroic reputation gain, wealth sacrifice) yields high health (0.99 probability), dominant 'Okay' wealth (0.90), elevated 'Hero' reputation (0.40), and near perfect safety (0.99), resulting in an uplifting finale focused on moral fulfillment and security, with no random peril events. Conversely, a bolder, self-interested path-searching the ridge (wealth opportunity, increased danger), accepting the favor (reputation shady tilt, risk up), rushing the cave (health risk, potential gain), and opening the chest (wealth boost, reputation penalty) shifts toward solid health (0.97), higher 'Rich' wealth (0.15) but 'Okay' dominance (0.84), neutral-to-shady reputation (0.78 neutral, 0.07 shady), and safe but vulnerable surroundings (0.97 safe, 0.03 threatened), triggering a "small luck" event like a coin gift while emphasizing material gains amid lingering uncertainty. These divergences underscore how belief propagation amplifies choice-driven multipliers, creating personalized, probabilistic narratives without fixed scripts.

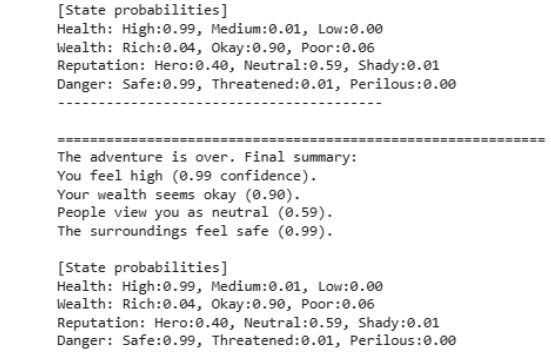


Fig. 8: Scene 2 where player does not get rewards for his choices [**?**].

D. Limitations

Despite strong performance on a small graph, the implementation has several limitations:

- **No learning capability:** Factors do not adapt across playthroughs, making behavior predictable.
- **Fixed graph size:** The design is tightly coupled to four variables, limiting scalability.
- **No graphical interface:** Console-only interaction restricts accessibility and engagement.
- **Static behavior:** Identical choices lead to identical marginals, reducing variability and replay value.

E. Future Work

Although the current version of the *Adventure Quest* game performs well and generates adaptive narratives using factor graphs, belief propagation, and evidence application, there remains significant scope for extension. Future enhancements can make the system more interactive, scalable, and adaptive, while enabling exploration of advanced probabilistic inference techniques and richer narrative structures. Key directions for future work are summarized below.

1) Graphical User Interface (GUI): The game currently operates through a text-based console, which limits user engagement. Integrating a graphical user interface using frameworks such as *Tkinter*, *Pygame*, or *Streamlit* would:

- Improve usability and interactivity.
- Allow players to select choices via buttons instead of numeric input.
- Visually display marginal probabilities, factor graph structures, and narrative branches.

2) Support for Larger Graphs: The current implementation is optimized for a small factor graph. Extending the model to include more variables and higher-arity factors would:

- Introduce a substantially larger inference space.
- Enable more complex and diverse narratives.
- Require optimization of belief propagation, potentially incorporating variational inference or Gibbs sampling to maintain scalability.

3) Adaptive Difficulty and Model Behavior: To improve replayability and challenge:

- Controlled noise or damping factors can be introduced into message updates.
- Learning-based mechanisms, such as Bayesian updates to factors across runs, can allow the model to adapt over time.
- Difficulty settings can simulate sub-optimal inference at lower levels to produce more human-like behavior.

4) Multiplayer and Online Play: A multiplayer mode-local or networked-could allow multiple players to influence a shared factor graph, with the inference engine acting as a collective narrative generator. Group play data could be used for AI training, and such a setup can be implemented with minimal computational resources, making it suitable for experimental and educational use.

5) Inference Enhancements: Inference quality and engagement can be improved by layering classic belief propagation with approximation and explainability techniques. Convergence speed can be enhanced using damping, dynamic scheduling, or residual belief propagation that prioritizes unstable messages. Anytime inference under fixed time budgets would allow partial marginals to be computed, supporting responsive interfaces and natural "thinking" pauses.

Sampling-based approaches, such as particle filtering, can enable support for continuous variables (e.g., a "luck" score). Beyond BP, learning-augmented inference can be explored by training graph neural networks through self-play to approximate messages, or by using variational autoencoders as prior generators. Variational inference methods (e.g., mean-field) can also be implemented and benchmarked against BP in terms of accuracy and speed.

To enhance narrative diversity, "mood profiles" can be introduced to modulate factors and sampling strategies, producing optimistic, grim, or unpredictable story styles. Lightweight explainability can further improve transparency by displaying brief rationales for belief shifts after each player choice.

6) Promising Research Directions: Future research can be organized around three complementary tracks: curriculum learning, explainability, and algorithmic trade-offs [10]. Curriculum learning can progressively train models on increasingly complex graphs, measuring transfer efficiency through convergence speed and stability. Explainability studies can evaluate player engagement and perceived coherence with and without probability visualizations. Finally, systematic benchmarks can compare loopy belief propagation, variational methods, and sampling-based inference across graph sizes and compute budgets, reporting accuracy, speed, and anytime behavior. Reproducibility can be ensured by fixing random seeds, standardizing priors, and publishing code and datasets.

V. Conclusions

This project successfully demonstrated a text-based adventure game that integrates probabilistic graphical models with interactive storytelling. Factor graphs were used to represent dependencies between game variables, while belief propagation enabled efficient approximate inference to generate adaptive narratives. Evidence multipliers further allowed the system to respond dynamically to player choices without requiring costly recomputation.

The game consistently produced coherent and responsive narratives, validating the effectiveness of belief propagation in small loopy graphs. Its modular design makes the system easy to extend, whether by adding new variables, introducing advanced inference techniques, or developing a graphical user interface. Overall, this work provides a strong foundation for intelligent narrative agents and demonstrates how theoretical concepts in probabilistic inference can be applied in an engaging, interactive setting.

References

[1] Wikipedia Contributors, "Factor graph," *Wikipedia*, Sep. 16, 2019.
[2] Wikipedia Contributors, "Belief propagation," *Wikipedia*, Jun. 17, 2020.
[3] "Sum-Product Algorithm in Probabilistic Graphical Models," upGrad Blog, Dec. 22, 2020.
[4] A. Madi, "Loopy Belief Propagation in Games," *Medium*, Jun. 02, 2023.
[5] S. J. Isenberg, "Message Passing and Normalization in Graphs," *Journal of Probabilistic Models*, vol. 12, no. 2, pp. 65–72, Apr. 2018.
[6] R. Korf, "Iterative Message Passing: An Optimal Approximate Inference," *Artificial Intelligence*, vol. 27, no. 1, pp. 97–109, Sep. 1985.
[7] L. V. Allis, "Solutions in Narrative Games and AI," Ph.D. dissertation, Vrije Universiteit, Amsterdam, Netherlands, 1994.
[8] M. Campbell, A. J. Hoane Jr., and F. Hsu, "Probabilistic Games," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 57–83, Jan. 2002.
[9] C. Browne *et al.*, "A Survey of Inference Methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
[10] D. Fudenberg and J. Tirole, *Game Theory*, 1st ed. Cambridge, MA, USA: MIT Press, 1991.
[11] Z. Zhang, M. H. Dupty, F. Wu, J. Q. Shi, and W. S. Lee, "Factor Graph Neural Networks," *Journal of Machine Learning Research*, vol. 24, pp. 1–54, 2023.
[12] H. A. Loeliger, "An introduction to factor graphs," *IEEE Signal Processing Magazine*, 2004.
[13] "Probabilistic Inference and Factor Graphs – DeepDive," Stanford DeepDive tutorial/overview.
[14] I. Schwartz, "Factor Graph Attention," in *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.
[15] C. Taylor, "Factor Graphs for Navigation Applications: A Tutorial," *ION Navigation*, vol. 71, no. 3, 2024.
[16] Factor Graph Optimization (FGO) overview — EmergentMind topic summary (2025).
[17] X. Xin et al., "A Comparative Study of Factor Graph Optimization-Based GNSS/INS Positioning," *Remote Sensing*, 2023.
[18] B. J. Frey, "Extending Factor Graphs so as to Unify Directed and Undirected Graphical Models," 2012.