Stories from the Bottom Up: Emergent Narratives with Composable Story Sifting Patterns

Qianwen Lyu
School of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
ql2n20@soton.ac.uk

David Millard
School of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
dem@soton.ac.uk

Nicholas Gibbins
School of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
nmg@ecs.soton.ac.uk

Abstract

Emergent Narratives (EN) are a popular approach in game design where stories naturally emerge from players' interactions with the game world. Story sifting is an EN technique where the player's attention is drawn to particularly interesting narrative patterns, but these story sifting patterns can be complex and difficult to create. To address this, we have developed an event-based rules system that supports composable patterns: building blocks that allow higher-level stories to be constructed from lower level ones. We demonstrate our approach through a simulation (of an alien invasion scenario) alongside a powerful incremental story sifter that curates compound events. We conduct agent ratio and stability testing to demonstrate the system's robustness, and a complexity test to show how the performance of the sifter scales with agent numbers and map size. Our findings show that the system remains stable throughout. Our work demonstrates that an event-based rules approach can effectively support composite patterns, ultimately enabling the curation of more complex EN stories.

CCS Concepts

• **Human-centered computing** → **Human computer interaction (HCI)**; • **Applied computing** → **Media arts**.

Keywords

Emergent Narrative, Story Sifting, Composable Patterns

ACM Reference Format:
Qianwen Lyu, David Millard, and Nicholas Gibbins. 2025. Stories from the Bottom Up: Emergent Narratives with Composable Story Sifting Patterns. In *International Conference on the Foundations of Digital Games (FDG '25), April 15–18, 2025, Graz, Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3723498.3723809

1 Introduction

Interactive digital narrative (IDN) plays a crucial role in game design, providing a narrative space in which gameplay can occur and motivating players with narrative goals and objectives. IDN comes in many different forms. *Designed Narratives* are non-linear experiences that are authored in advance, and which can take a variety of structural shapes [13], they allow for interactivity whilst maintaining designer control over agency and are therefore a very popular approach that are used in many mainstream narrative games (for example, the works of Telltale, or in AAA adventures such as Ubisoft's Assassin's Creed series). *Generative Narratives* are not pre-authored but rather are created during play; they allow for wider agency, but provide only indirect control by the game designers [7, 19]. Finally *Emergent Narratives (EN)* do not impose narrative direction but rather allow stories to emerge through a player's interaction with a complex game world and rely on players' making sense of that interaction [29]; they allow maximum agency but offer the least control of the stories and their quality, leading some to describe them less as games and more as narrative instruments [17] .

Emergent Narratives are considered special by those that play them because they offer a unique and unpredictable experience that does not rely on pre-defined stories, and which feels personalised to the player [26]. Popular examples include The Sims [23], Rim World [27], and Dwarf Fortress [25]. These games offer huge narrative variety but at the risk that few interesting stories will emerge, or that those that do might be of poor quality, or buried in thousands of meaningless events, and therefore missed by players [26].

Game designers have a number of strategies to prevent this. For example, by setting up evocative narrative elements, or perturbing a stable simulation world through provocative events (for example, in Rimworld – depending on your choice of narrator – your space colony might encounter things such as cargo pods, solar flares, or pirate raiders [18]). Researchers have also looked at this problem, and suggested Story Sifting as a potential solution [26]. A Story Sifter scans system events looking for interesting narrative patterns; this might be *retrospective* (such as analysing a game log), or *incremental* (in which case it runs in real time and looks for patterns as they develop) [14]. In both cases it can then draw the player's attention to the most narratively interesting events.

A challenge for story-sifters is that sifting patterns are complex and difficult to author, leading to a limited set of stories that can be identified. In our work, we address this challenge by proposing the idea of using a rule-based story sifter to support *composable story sifting patterns*, which can be combined to generate more complex stories. In our model, low-level events in the simulation trigger higher-level patterns that are fed back into the simulation and can then trigger further patterns. We argue that these composable patterns more easily enable complex patterns to be built and that a rule-based story sifter is well-suited to support this process.

FDG '25, Graz, Austria

ACM ISBN 979-8-4007-1856-4/25/04
https://doi.org/10.1145/3723498.3723809

To demonstrate this idea we have designed a game simulation environment where the stories can unfold. The advantage of this approach (rather than a playable game) is that our simulator can rapidly generate large amounts of data for experimental purposes without needing human participants. However, our sifter follows the incremental approach, meaning that it operates at runtime and therefore could be applied to a live game environment.

The rest of the paper is structured as follows. Section 2 describes the history of emergent narrative games, their unique aesthetics, and prior work on story sifting. Section 3 presents our simulation, based around an alien invasion scenario, and describes the agents, environment, and interactions. Section 4 describes our rules-based story sifter and shows how patterns are defined. Section 5 evaluates our system in terms of its stability and performance. Finally Section 6 summarises our key findings, and discusses as future work how this type of story sifting might be coupled with a drama manager, before Section 7 concludes the paper.

2 Background

EN was first described by Galyean, who suggested that narrative structures might emerge from the environment as a product of interaction when players navigate a game experiences [9]. Galyean was setting EN up as a default way of experiencing narrative in games, rather than as a genre of narrative game in its own right, thus setting the scene for Designed and Generative Narrative approaches (what he calls "Narrative Guidance" [8]) that have since become the norm in popular narrative games.

EN as a distinct approach to narrative games was first explored by Aylett who described it as a bottom-up structure where the narrative emerges through interactions [1]. Aylett turned to EN as a method of addressing the *narrative paradox*, the challenge of maintaining narrative coherence while allowing users to be participants rather than mere spectators [2]. This view of EN is reinforced in more recent work by Ryan, who describes it as a bottom-up approach where narratives emerge out of a computer simulation of characters' activity [26].

This tradition of EN emphasises the simulated story world, and Ryan goes as far as suggesting that they need not even be interactive. An alternative view is set out by Mateas, who focuses on the player's role in constructing a narrative from their experience [20]. Walsh describes this view succinctly while reflecting on how players formulate narratives in games like The Sims: "In doing so you are making sense of events just as we make sense of events elsewhere in life; you are creating a narrative, not remediating an interactive narrative emerging from the session." [29].

Kreminski highlights this difference in emphasis (storyworld vs player) in their excellent discussion of historical interactive EN [16]. In our work we tend towards the simulated storyworld view, focusing on agent interactions within a simulation – although our approach certainly does not preclude a player, and could be targeted to involve them explicitly (see Section 6).

Almost from its very beginnings, the shortcomings of EN were acknowledged. EN is a bottom-up structure where authors create only possibilities. This means that the events generated by the simulation could be entirely random, potentially leading to stories that are boring, contradictory, or untellable. In her game FearNot, targetted at anti-bullying, Aylett acknowledges these problems and introduced a stage manager, a system that sets up scenes and characters and can even intervene to ensure that stories progress in certain ways [3]. Modern EN games often use different interventions in order to increase the probability of interesting events, Rimworld even incorporates this into play by offering three different storytellers, each representing a different level of difficulty, shaping the pacing and intensity of the player's experience by selecting different types and frequency of event [27]. Even The Sims manages narrative progress indirectly. Characters have hidden desire trees, where their goals progress from an early root goal to different final outcomes (for example, desiring a telescope as a child, to applying to be an astronaut as an adult) [4].

Drama managers are systems that take this intervention to extremes. A drama manager is an omniscient system that guides and shapes the story dynamically as players explore the game world [28, 30]. It continuously monitors the simulation and make decisions [21]. Façade[22] is an example of using a drama manager to guide the story's progression in a meaningful direction. It relies on *Story Beats* which are a collection of interactions. The drama manager dynamically sequences interactions based on the current narrative state, ensuring that the story progresses coherently.

Drama managers are typically seen as an approach in Generative Narrative rather than EN. Ryan argues that any intervention risks compromising the unique aesthetics of EN by making the story more likely to appear well-formed rather than spontaneously emerging from the simulated story world [26] risking key aesthetics such as the thrill of experiencing actual, improbable, and unauthored events.

Ryan suggested an alternative approach to curating ENs. He initially termed this "story recognition", suggesting the possibility of developing a system capable of retrospectively selecting the interesting stories from a simulation log. Later he introduced the term "story sifting" to describe the fundamental task of selecting compelling stories from a morass of events [26].

Research in this area has been growing, for example *Felt*, a query language-based story sifter [15]. Felt allows for the creation of complex sifting patterns rather than relying solely on literal sequences. This approach enables story sequences to be non-continuous, allowing for the inclusion of other unrelated events interspersed within the narrative. In this system, events that emerge from the simulation are stored in a database, and users create sifting patterns using a query language. These sifting patterns consist of a set of logic variables and their relationships. The logic variables can be substituted with specific types of database entities, such as characters and events, while the relationships define the constraints on the values that these variables can take.

Felt has a few shortcomings. Firstly, the sifting pattern authoring might be difficult since the story sequence can be long. Although query languages are flexible, it is still a burden for users to write and maintain. Additionally, this work is retrospective, and all the events were recognized after they emerged from the story world. This means they could not be incorporated into a live game. To solve these issues, Kreminski introduces a domain-specific language Winnow and the idea of *incremental story sifting* [14]. In Winnow the sifting patterns try to match a series of ordered events, disregarding irrelevant events. To make it incremental the system creates

a partial matches pool to save sifting patterns and branches. Each time a new event occurs, the system compares it with the sifting patterns in the pool. The event only needs to be a partial match rather than a perfect one to be stored.

Clothier and Millard use an incremental story sifter to create a drama manager that they argue does not disrupt the aesthetics of EN [5]. In their game, Awash!, the system first passes each event to an incremental story sifter to identify emerging patterns. These unfinished stories are then passed to a drama manager which indirectly intervenes in an attempt to increase their probability of completing. Their results show how incremental sifters might be used in games to improve narrative quality whilst limiting any impact on EN aesthetics.

This prior work shows that story sifting is effective and can be used in game environments at runtime to both draw a player's attention to interesting stories, and to drive possible interventions. However the problem remains that sifting patterns are complex to author. Johnson-Bey and Mateas also view patterns as sophisticated queries (as in Felt) but they present *Centrifuge*, a visual tool for constructing sifting patterns [12]. Centrifuge allows authors to create patterns in a node based drag and drop environment, with queries, logic, and functions represented as special kinds of nodes. They demonstrate their approach in a simulated game called "Talk of the Town" a social simulation that models businesses, occupations, and relationships. They identify pattern re-use as an important missing feature that would create modularity.

Our approach is similar to Felt and Centrifuge in that we represent sifting patterns as queries that match against the game world. However in our work we represent that world as a series of events, and that enables us to define patterns as sequences of events, and by reinserting completed patterns as new high-level events, we enable composable patterns and thus modularity. We are also inspired by their methodology, creating a game simulator that enables us to easily test our story sifter.

3 Simulation

The first step in exploring composable patterns is to build the game simulator that can act as the basis of the project: a story world in which stories can unfold. There are a few advantages of building up a simulator rather than a game. Most significant is that the simulator can be run many times without human involvement, and thus generate a large amount of data that is highly beneficial for testing. There is a secondary benefit in that a simulator does not require a player interface that acts as an intermediary. This significantly reduces the development burden, but also removes the quality of that interface as a variable when testing.

Figure 1: The architecture of the project

Fig 1 shows the architecture of the whole system. It is made of three components. The *Simulator* generates events and pass them to the *Story Sifter* while the simulator is running. The Story Sifter curates the potentially compelling parts of the stories by comparing those events to a set of stored patterns. When a pattern is complete, the story sifter generates a high-level event to summarize the story and then passes it back to the simulator. The high-level event is regarded as a new event and can be responded to by agents in the simulator, and in turn further curated by the story sifter. It is in this way that patterns can be composed from other patterns. The whole interaction of the Simulator and Story Sifter are logged, this log is then passed to a *Visualizer* where the events can be replayed, and the stories that were identified explored.

The simulator is designed to be an event-based system. Inside the simulator are many agents who have goals and can do actions based on their environment. We define an event as a change of state [10], which could be in the simulated environment (such as the movement of an agent from one place to an adjacent place) or in the agents themselves (as a transition in their behavioural state model). When a state changes, the simulator generates an event with a series of parameters including a time stamp, the content of the event, and an event ID.

3.1 Scenario

In order to explore story sifting patterns we need to have a story world, we therefore need to come up with a theme that can generate lots of dramatic conflicts. In 'Talk of the Town' Johnson-Bey and Mateas used a social scenario that maps well to social management games such as The Sims [12]. We have chosen a more melodramatic scenario that is closer to the strategy genre, and modeled an alien invasion scenario in a small town (inspired by popular films such as Independence Day, War of the Worlds, and Mars Attacks).

The simulator was developed in JavaScript. Within the simulator, there are three types of character agents: townsfolk, aliens, and soldiers, each of which will be dynamically spawned on the map and move freely. As time passes by, agents may encounter each other and interact. The town map is a grid of cells and is procedurally generated by code, incorporating randomized roads and buildings. In this section, we will discuss the simulator in detail from two aspects: the agents and the map generation.

3.2 Agents

Each agent runs a Finite State Machine (FSM) that dictates its behaviour, they receive signals from the environment, and can take actions depending on their current state. The FSM is based on a set of states-condition-action. The action can be an interaction with the world or other agents, or it could be an internal state change. An event will be generated when the state changes, and recorded into the events log. For example, a townsfolk may begin with a "wander" state, but will switch to a "run away" state when encountering an alien nearby. This state change can be recorded as an event labeled "A runs away from B" where A and B are identifiers.

There are three types of agents with different FSMs: alien, soldier, and townsfolk:

- **Aliens:** Wander around the town, prioritizing the destruction of buildings and attacking human characters (soldiers and townsfolk).

- **Soldiers:** Patrol the town, protect townsfolk, and attempt to kill any alien invaders they encounter.
- **Townsfolk:** Walk around or hide in buildings. Running away if they see an alien nearby.

Fig 2 shows an example of a soldier's FSM. Soldiers can be in one of five states: patrol, run away, chase, attack, and died. They usually start in the patrol state. The states can transfer to others based on some conditions. For example, when a soldier sees an enemy: if the enemy is out of attack range, they will chase the enemy; if the soldier is low on health they we will run away from the enemy; otherwise, they will attack the enemy. The attack state may later transfer to the chase state if the enemy escapes out of attack range, but still remains in visual range.

3.3 Map Generation

There are lots of popular approaches to generate random indoor or outdoor maps, like Binary Space Partitioning [24] for rooms or the Diamond Square Algorithm [6]. In this project, we used the L-system algorithm, since this algorithm can be applied reletively easily to a town. An L-system is a parallel rewriting system that consists of an initial axiom which is the initial string, an alphabet which indicates all the symbols in this system, and a set of production rules that define how the variables in the alphabet could be replaced [11]. It starts with the axiom, and then in every iteration, replaces the variables with the rules.

In this project, we defined the L-system as follows:

$$axiom : [F] - -F \tag{1}$$

$$alphabet : \{F, [,], +, -\} \tag{2}$$

$$rules : F- > [+F][-F], F- > [+F]F[-F], F- > [-F]F[+F] \tag{3}$$

"F" means to move forward by one unit. "+" means to turn right for 90 degree, while "-" means to turn left for 90 degree. "[" means to save the current state, and "]" means to pop state. In order to make the roads look more natural, we used three rules to replace the variable "F". In each iteration, we will randomly select one rule for each single "F". We also added a probability to ignore rules, making the roads more irregular. After a few iterations, we got the final string, which can used to draw the random roads of a city. For example:

- iteration1: [F]–F
- iteration2: [[+F][-F]]–[+F]F[-F]
- iteration3: [[+[+F]F[-F]][-[-F]F[+F]]]–[+[+F][-F]][+F][-F][-[-F]F[+F]]

Once the roads are generated, we add buildings by traversing those roads, and then generating one randomly sized building on each side of this road (checking for clashes with other buildings).

Figure 4 presents an example of the visualizer. On the left side, you can see an example of a town map generated with this method, with gray roads and orange buildings providing a clear layout of the environment. Character interact with this environment in a number of simple ways, townsfolk can hide in buildings, buildings block lines of sight and movement, and all agents move more quickly on roads than land.

4 Story Sifter

The story sifter is the central element of our research. It uses a set of sifting patterns to curate compelling stories from the vast number of events generated by the simulator.

4.1 Incremental Story Sifter

As we discussed in Section 2 strictly speaking we are developing an incremental story sifter that is capable of story sifting while the simulator is running. It therefore has a sense of both patterns that have completed as well as those that have started and have the potential to complete. Similar to Winnow [14], we use a pool to store partial matches, which represents the potential stories. When a new event occurs, the story sifter performs two tasks:

Initiating New Partial Matches: The sifter checks all of its story patterns to see if any new patterns can be initiated by the new event. If it does, then the story sifter will initiate the partial match and put it into the pool. But if there is already one partial match with the same events and same pattern in the pool, it will be skipped to avoid repetition.

Checking Existing Partial Matches: The sifter then checks the pool to see if the new event progresses any existing partial matches. If it does, it adds the event's ID to this partial match's event list. The sifter then checks if the partial match is complete. If a partial match is completed, it will be removed from the pool, and a high-level event representing that pattern is generated. This high-level event is then added back into the simulator, making it available as part of other patterns.

4.2 Patterns

In describing the patterns it is helpful to distinguish between low-level events (which are natively generated by the simulator – for example, an agent moving or attacking another agent) and high-level events (which are generated as a result of the completion of patterns). Patterns can be defined using either type of event.

In this project, we defined five high-level events: "almost kill", "escape", "defeat", "giant killer", and "lucky kill". We then defined two additional high-level events that draw on these, "revenge" and "ace killer". Although the system does not distinguish them we call these last two story-level events, and can monitor them separately as they are the composite patterns that players would be most interested in. As an example of how these composable patterns work we can look at the "revenge" pattern. A "revenge" describes a story where A attacks and almost kills B, then B escapes, and later B comes back and kills A. It requires at least three high-level events: "almost kill", "escape", and "defeat". Each of these high-level event needs a few low-level events. In this case, an "almost kill" event includes "A attacks B", "B is badly hurt", and then "B runs away from A". An "escape" event includes "A run away from B", and "A successfully run away from B". A "defeat" event includes "A attacks B", and "A kills B".

We defined patterns in a JSON file. The pattern has a condition list describing the sequence of events that make up the story and whether these events should be repeated. Figure 3 shows the "lucky kill" pattern, where one character attacks and immediately defeats another.

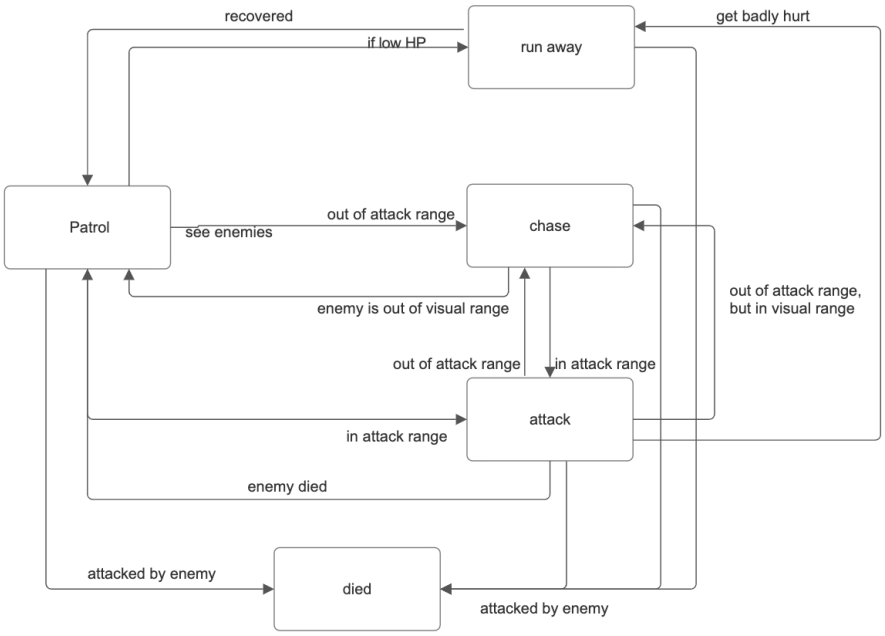


Figure 2: The soldiers' states diagram

- The "time-limit" indicates that if a partial match initiated with this pattern exceeds 20,000 simulation time units without completion, it will be terminated and removed from the pool. In this case there is no strict need for a time limit (given the current actions available to the agents) so we set this to a high value.
- The "unless" condition specifies situations that would terminate and remove the partial match from the pool. In this case if the target dies the pattern can never be fulfilled and is removed.
- The "unless-forever" is a special case that if the event in this list happened, the partial match will be labeled false and left in the partial match pool forever. No new partial match of the same type with the same characters will be initiated (if this case if A attacks B and kills B at the very first attack, then it is a lucky kill. Otherwise, A will never have another lucky kill to B.).

```
"lucky_kill": { "events": [
{
   "char1Idx": 0,
   "tag": "attacked", "char2Idx": 1, "repeat": false
}, {
}
   "char1Idx": 1,
   "tag": "was killed by", "char2Idx": 0, "repeat": false
}
], "unless": [
{
   "char1Idx": 0,
   "tag": "was killed by" }
], "unless_forever": [
{
   "char1Idx": 0,
   "tag": "attacked" }
],
"tag": "luckily killed",
"type": "high-level",
"main_characters": [0, 1]
}
```

Figure 3: 'Lucky Kill' Pattern

4.3 Visualizer

The simulator requires a visualiser that converts the event log onto a visual representation that is easier to understand and follow. Figure 4 shows the visualiser interface. The map is displayed on the left, with roads in grey and buildings in dark orange. Characters are represented by squares labeled with their names: aliens are red, soldiers are dark blue, and townsfolk are green. When a character dies, their square turns grey.

On the right side, there are two scrollable views. The left scrollable view lists all the successfully curated high-level events. When the "show" button in a high-level event is clicked, the right scrollable view displays the low-level events that constitute the high-level event. This also causes the map to jump to the frame where the high-level event was happening. At the bottom of the interface

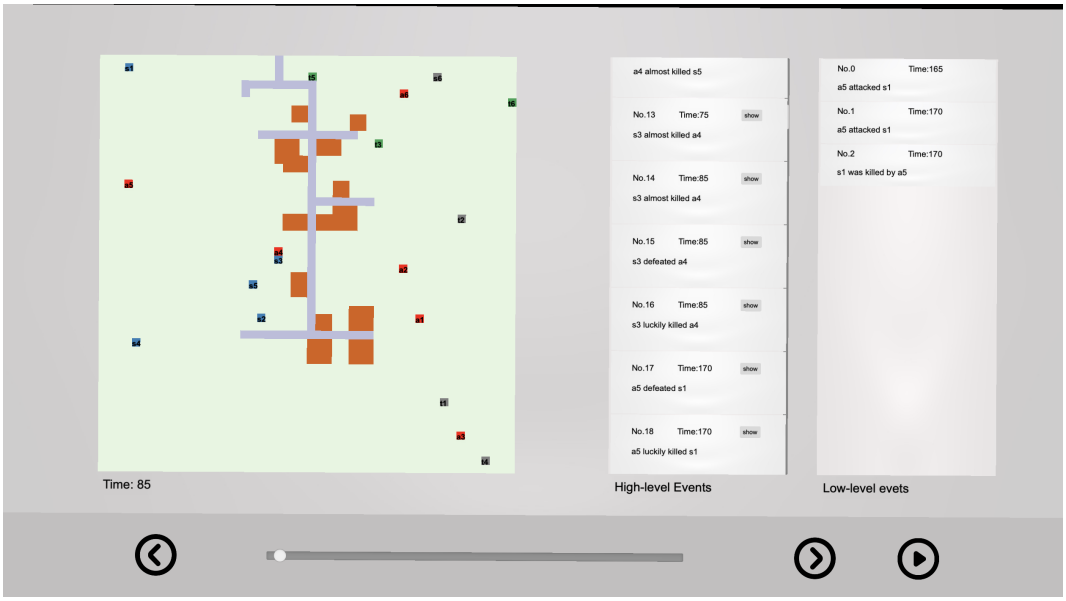


Figure 4: An example of the visualizer

is the controller part, including play, forward, and backward buttons, along with a slider. These controls can be used to manage the playback of the run.

The visualizer requires several inputs to accurately reconstruct these events:

- **Map data:** A description of the town map marking roads and buildings.
- **Events log:** A log recording all event information in the form: character 1, event, character 2, time, and ID.
- **States log:** A log recording all the state information of the objects in the simulator. It includes the object name, state, position, and time.

The states log is there for performance reasons. Initially it is required for the starting state of all of the agents, in theory the state of the simulation at any given time could then be generated by running the events from the beginning, however this makes it expensive to jump between times. The states log not only holds the state at time 0, but also records it every 1000 time intervals - thus when jumping to a new point the simulator doesn't have to run the simulation from the start every time.

5 Evaluation and Performance

Our goal with this research is to find out how a rules-based story sifter can support composable patterns. In the previous sections, we demonstrated how this system was designed and constructed. In this section, we will analyze and evaluate the system to demonstrate that it works as planned. To achieve this, we firstly investigated what would make a suitable ratio between the character types. Then we conducted a complexity test to assess the system's performance as the number of agents and map size changes. Finally, we conducted a stability test to ensure the system behaves predictably enough for further study.

5.1 Methodology

To clarify the evaluation process, several key terms are defined below:

- **Partial Match:** The story sifter compares each new event from the simulator with all predefined patterns. If the event matches the first event of a pattern, a partial match is generated and placed in the partial match pool.
- **Completed Match:** If/once all the events in a pattern are matched, this pattern is completed. It will be removed from the partial match pool and generates a new event.

Partial Matches and Completed Matches refer to all seven of our patterns. Five of these are high-level events and two are story-level events, and this gives another way to monitor how the story sifter is performing. Put another way, the number of partial matches is the sum of initiated high-level events and initiated story events, while the number of completed matches is the sum of completed high-level events and completed story events.

Before running our tests we first need to find a suitable character ratio (between townsfolk, soliders, and aliens), since different character ratios might produce different results. We explored ratios in the range 1 to 3. We aim to avoid any one type becoming dominant, as this would lead the simulator to equilibrium too quickly (for example, if the aliens kill all other characters then no more interactions would emerge). We therefore chose to test the ratios: 1:1:1, 1:2:2, 2:1:2, 2:2:1, 1:1:3, 1:3:1, and 3:1:1. The ratios are represented as [alien: soldier: townsfolk].

We used the number of partial matches initialized as an indicator of story richness. We ran the simulator with our different ratios and analysed the data. Two major factors were considered in choosing the most suitable ratio:

- Generates a sufficient number of partial matches.

- The number of partial matches generated does not vary significantly.

For each ratio, we ran the simulator one hundred times to obtain the average number and the variance of the partial matches created. In all cases the simulator was allowed to run for twenty thousand time cycles. We then created plots to compare the results.

For the complexity test, we aimed to determine how the number of characters and the map size affect the simulator's execution. We selected three character counts (total number of agents): 20, 50, and 100, along with four map area sizes: 2500, 12500, 22500, and 40000. We tested the map sizes paired with the three character counts in the best ratio from the previous experiment, resulting in twelve combinations. For each combination we ran the simulator 100 times and recorded data of the execution time, the total events, initiated partial matches, completed matches, initiated high-level events, completed high-level events, initiated stories, and completed stories. We then analyzed these data trends.

For the stability test, we aimed to find out the variance in results when running the simulation multiple times. Ideally we are looking for a range of results, but where the extents of that range are not wildly different. We used the same ratio as before, with a map size of 10,000 and 50 agents. We checked for crashes or errors during the simulator's execution and calculated the variances of the execution time and partial matches to see how the numbers vary when map size and character size changes.

5.2 Ratio Selection

Firstly, we compared the different ratios from several aspects including partial matches number, partial matches variances, initiated stories number and completed stories number. Fig 5a demonstrates the average partial matches generated (based on one hundred runs each). Ratio 2:2:1 generated the highest number of partial matches of over 450, while the ratio 3:1:1 and 1:1:3 produced the lowest of below 300. Ratio 1:1:1 generated the second highest number of partial matches of around 400.

Fig 5b shows the variance of each ratios' partial matches generation. A higher variance value indicates that for a given ratio, the number of partial matches can vary significantly across different runs. As we can see, the variance of ratios 1:3:1, 3:1:1 and 2:2:1 are high (over 4000). Whereas we aim to keep the variance relatively low so that there is some consistency between runs.

Lastly, we compared the number of initiated and completed stories, as shown in Fig 5c and Fig 5d. Ratios 1:1:3, 3:1:1, and 2:1:2 performed poorly in initiating stories (under 25), while the other ratios generated an average of over 33 initiated stories. For completed stories, most ratios achieved an average of over 6.5. However, ratios 3:1:1 and 2:1:2 had the lowest performance, with fewer than 5 completed stories.

When choosing a ratio to use for the rest of the experiments we are looking for something with high partial matches and stories and low variance. 2:2:1 performs the best on initiated matches and stories but shows high variance. Overall, the 1:1:1 ratio appears to be the best choice, offering the second highest partial matches and completed stories coupled with relatively low variance. Therefore, we selected this ratio for our subsequent research.

5.3 Complexity Test

Having chosen a suitable ratio we now look at how the map size and character number impact the execution time and pattern matching.

From Fig 6a, we can see that the execution time nearly doubles when the characters size doubles, but when the map area size increases, the trend of execution time rise more gradually. Fig 6b illustrates the average total events. As the number of characters in the simulator increases, the number of generated events also rises. However, when the map area size changes, the total number of events initially increases (more space initially means that the agents survive longer, leading to more events) but tends to level off after reaching the size of 22,500 (as the number of interactions decreases due to fewer encounters in the larger area).

Fig 6c and Fig 6d shows that both partial matches and completed matches follows a slightly different pattern, dropping as the map area increases even at the smaller scales. Although as before the matches increase proportionally with the character size.

Finally, we compared the trend of the high-level events and stories separately. From Fig 6e and 6f we can see that approximately one in four initiated high-level events reach completion. Similar to partial matches, the number slightly decreases when the map area size increases, but grows proportionally with the character size.

Fig 6g and Fig 6h shows the initiated and completed stories (composite patterns). Approximately one in five initiated stories are completed which is only slightly lower than the other patterns. As before the number slightly decreases when the map area size increases, but grows proportionally with the character size.

With regard to these complexity test, we can draw a number of conclusions. Firstly, on a larger map, fewer meaningful events are generated. This is probably as it becomes less likely for characters to meet when the map size grows, which results in fewer interactions. Secondly, when there are more characters in the simulator, the number of events, partial matches, high-level events, and mini stories grows significantly as the opportunity for interactions increases. However, it is clear that agent count has a more significant impact on interactions, as when we double the number of agents from 50 to 100 and (approximately) double the map size from 22500 to 40000 they do not cancel each other out and we still see a significant increase in interactions (events and matches). This is probably because the agents do not rely purely on chance to encounter each other but actively pursue one another, reducing the impact of scale.

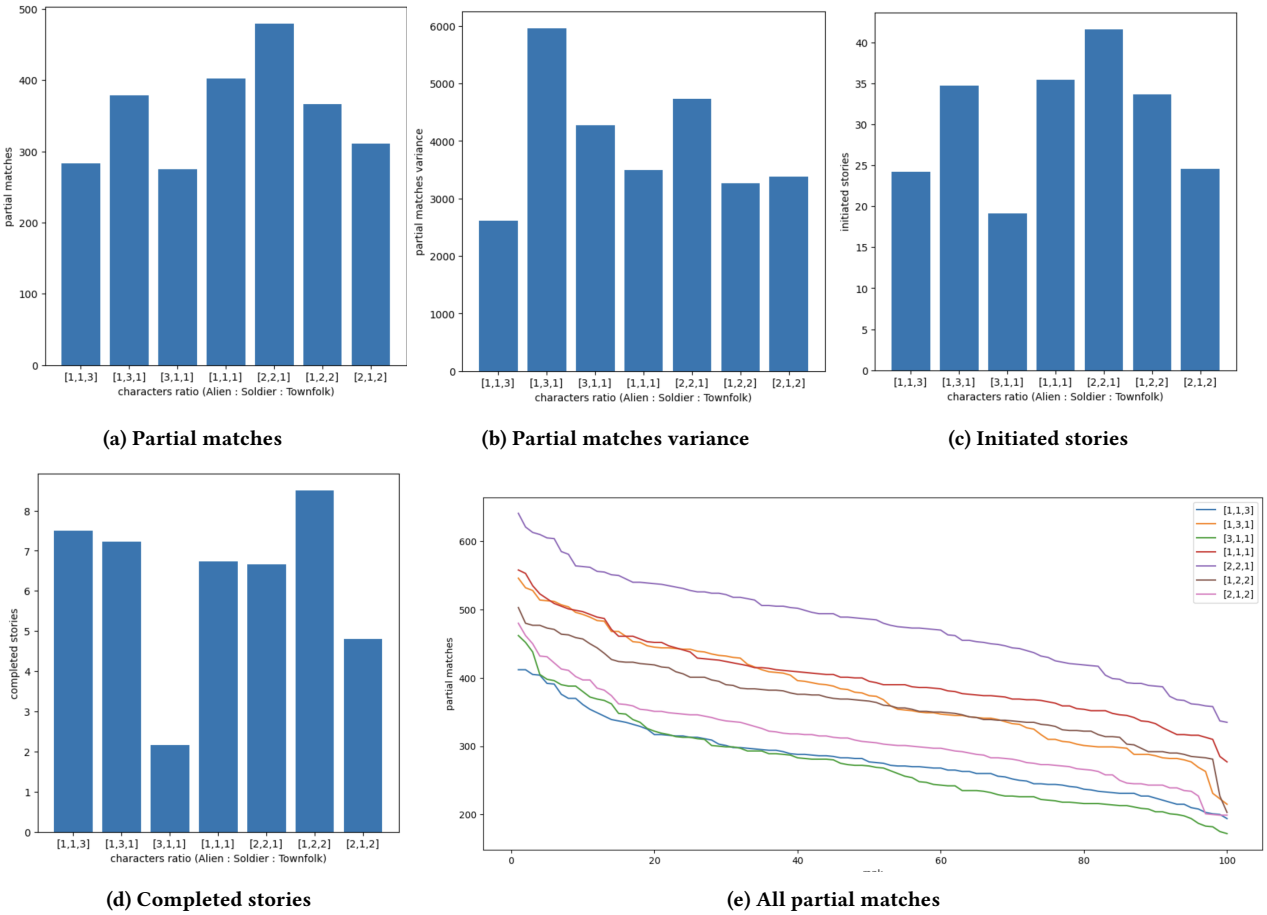
5.4 Stability Test

Our final analysis was to look at how our system behaves across multiple runs. A stable system would show a relatively small variance with few spikes.

Each of our twelve scenarios were run one thousand times and the differences explored.

Fig 7a and Fig 7b illustrates the execution time over those one thousand runs. We can see that most execution times converge in a range of approximately 2,000 to 3000 milliseconds. Only a few runs have an unusually high or low execution time. There are no crashes or errors during the runs.

Fig 7c and Fig 7d demonstrates the generation of partial matches and completed matches. We can see that most partial matches are in a range of 350 to 450, while the completed matches are in a



(a) Partial matches

(b) Partial matches variance

(c) Initiated stories

(d) Completed stories

(e) All partial matches

Figure 5: Ratio selection

range of approximately 100 to 150. We then calculated the variance of the execution time with three character sizes: 20, 50, 100, and two map area sizes: 10,000 and 20,000. We can see in Fig 7e that with the increase of the number of characters, the execution time variance increase significantly. However, the map area size have little influence on the execution time.

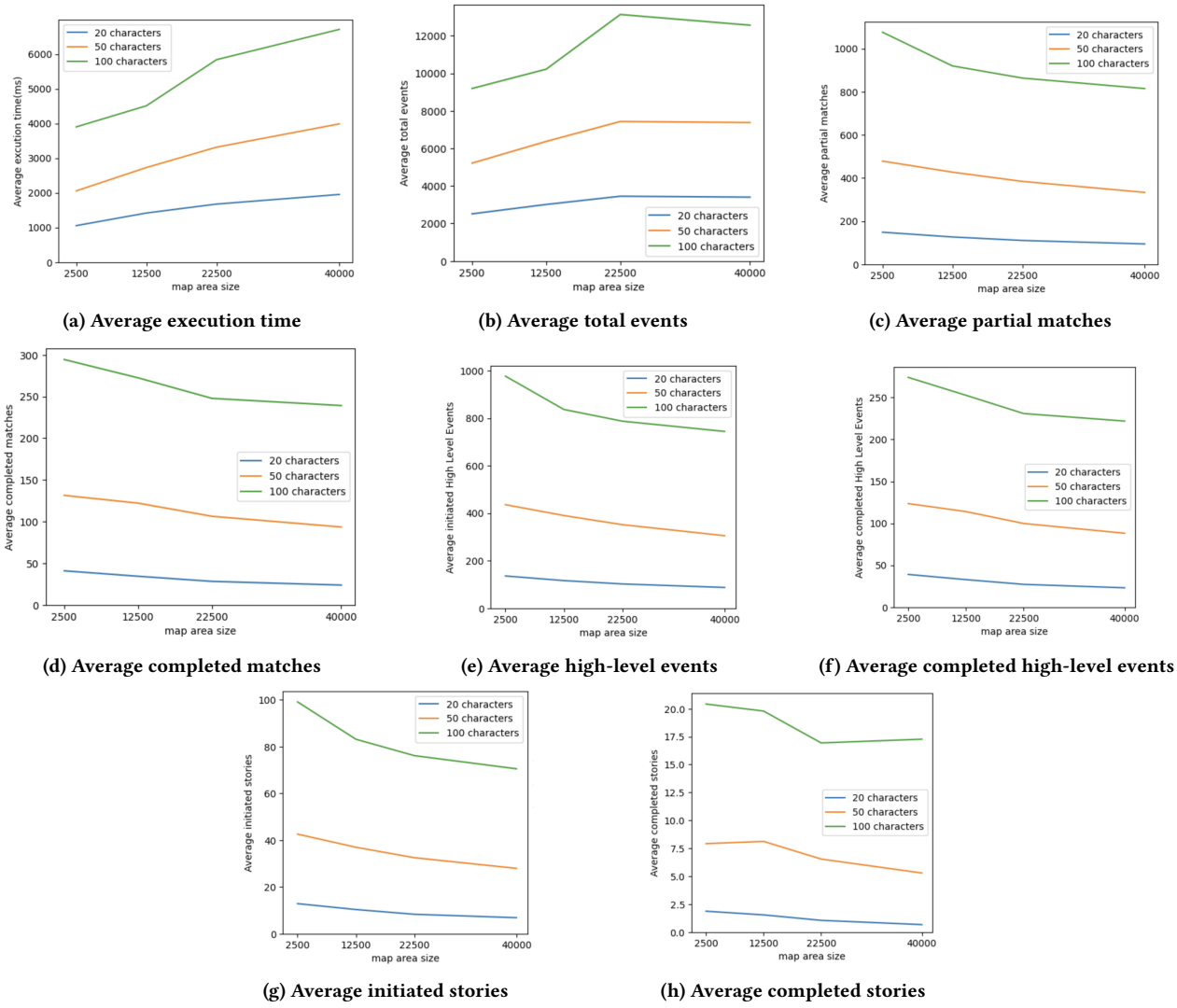
In Fig 7f, we calculated the variance of the partial matches with three character sizes: 20, 50, 100, and two map area sizes: 10,000 and 20,000. Similar to Fig 7e, the map area size has less effect on the variance, while character size plays a more important role.

From the stability test, we can see the following points. Firstly, over 1000 runs, the system seems to be robust, since there are no crashes, errors, or significant changes. Secondly, the map area size has little influence on the variance in execution time, indicating that the simulator tends to use a similar amount of time for a run regardless of the map size. Thirdly, similar to the execution time, the number of characters is more influential on the variance in pattern matches than the map area size. So the more characters in the simulator, the more likely for the output of the system to vary.

6 Discussion

Our work builds on key research in story sifting, especially prior work on real-time story curation [14] and story sifter intervention within games [5]. Previous work has also explored the challenges of defining story sifting patterns, established game simulation as an appropriate methodology, and identified modularity as a key research challenge in the space [12]. We address this challenge directly, with the goal of establishing a game simulation framework to demonstrate and test composite patterns. In our data, the "completed stories" represent these composite patterns. Our contribution lies in the provision of an event-driven game simulation, and demonstration that composite patterns can be supported by reinserting completed patterns as new events in the simulation. This enables completed patterns to be reused as components of more complex patterns while simultaneously reducing the complexity and overload of managing multiple patterns.

To show that the story-sifter is effective in this context we have performed stability tests to show that the behaviour is within an acceptable variance between different runs, and explored how the complexity of the simulation (in terms of number of agents and map size) impacts performance. Predictably, both execution time



(a) Average execution time

(b) Average total events

(c) Average partial matches

(d) Average completed matches

(e) Average high-level events

(f) Average completed high-level events

(g) Average initiated stories

(h) Average completed stories

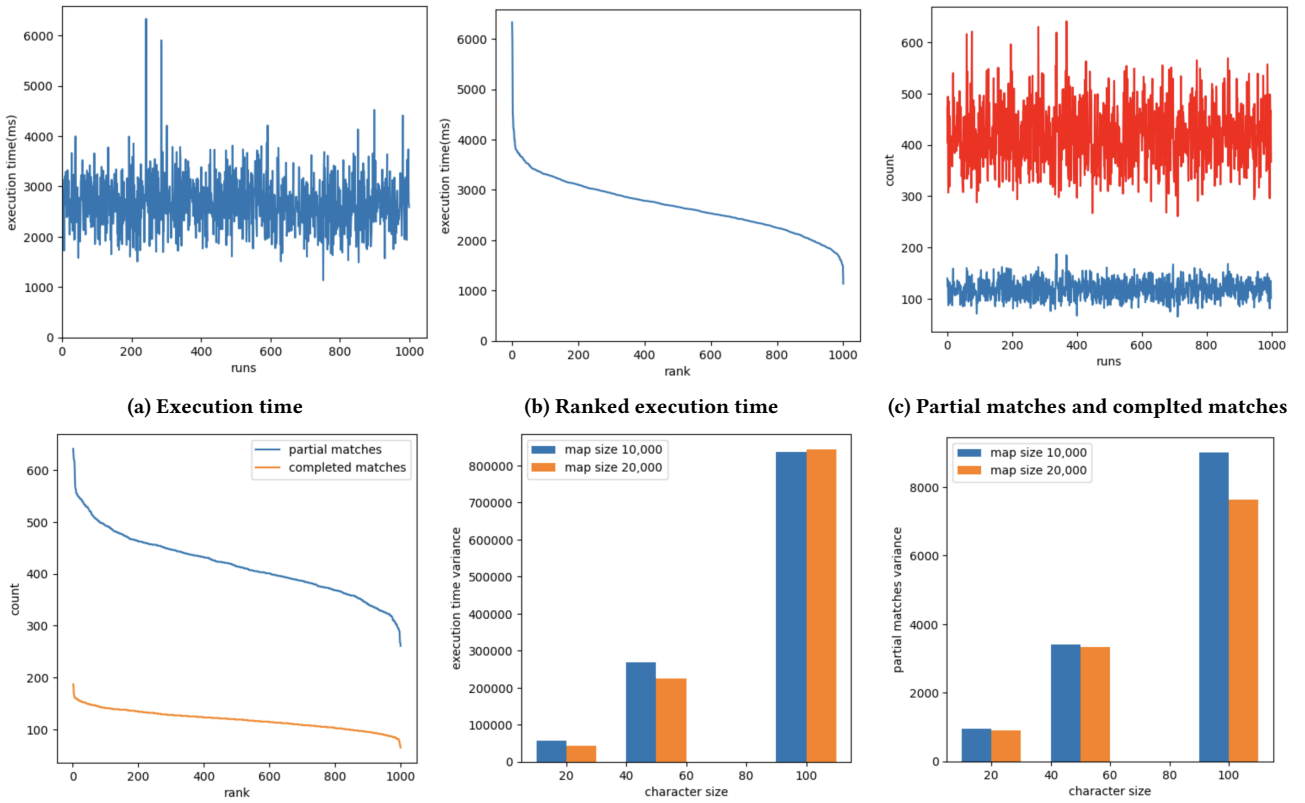
Figure 6: Complexity Test

and the number of matches increase significantly as the population grows. More agents (characters) in the system also leads to a greater number of events and stories - which is what we would expect to see if the simulation was running correctly. Map size has less impact, initially leading to slightly more events (as survivability increases) and then as the map size grows significantly it leads to fewer events (as interactions decrease) although these seems to be mitigated by the characters behaviour (in actively pursuing one another).

Overall, the evaluation shows that the system effectively supports composable patterns and scales well to higher complexities. One limitation is that we have a relatively small number of patterns and did not test the impact of alternative numbers of patterns on execution time, this is a clear direction for future work. Another avenue would be to explore user testing to explore to what extent composable patterns and therefore modularity impact authoring (as suggested by Johnson-Bey and Mateas [12]).

Beyond these extensions to our existing study we also hope to look at how a protagonist may be factored into our simulation. At present the simulation assumes an omniscient player that can see all events equally, this makes it a good simulation of certain game genres where this is the norm (e.g. large scale strategy games, or management simulations) but not for games where the player has an avatar in the game, and sees the game world from that avatar's perspective. Explicitly modelling one of the agents as a protagonist would solve this issue, and would also provide the possibility of focusing the story sifter on events local to the player, and ignoring remote events - which might have a positive impact on performance.

Finally we are eager to explore how our simulation environment might act as an alternative way to explore story sifting intervention in a quantitative manner. Clothier and Millard describe three forms of intervention that could be used to make stories more likely to complete [5]: *generative*, where the simulation is altered without



(a) Execution time

(b) Ranked execution time

(c) Partial matches and complted matches

(d) Ranked matches

(e) Execution time variance with different character size

(f) Partial matches variance with different character size

Figure 7: Stability Test

regard for the rules of the simulation (e.g. inserting new agents, or moving agents instantly from one place to another); *direct*, where the intervention is always within the rules of the simulation (e.g. forcing a state change within an agent); and *indirect*, where the system merely adjusts probabilities to make stories more likely to occur. We would add the possibility of indirect *player intervention*, where the player is nudged rather than the agents in the simulation (e.g. by creating additional goals, or just making suggestions).

Direct and indirect interaction could be explored within the scope of our existing simulation, and generative and player intervention could be explored with the addition of a protagonist as described above (direct intervention should be beyond the player's sphere of knowledge to maintain the illusion of a logical world, and player intervention could be modelled by adjusting the behaviour of the protagonist agent rather than other agents). Our event-based game simulation thus provides a robust platform for exploring the impact and behaviour of these alternative approaches in the future.

7 Conclusion and Future Work

In this paper, we have presented an event-based approach for Emergent Narrative using composable patterns, this includes a game simulation that generates events, an incremental story sifter that curates interesting stories in real time, and a visualizer to review events within the simulator.

Building on this foundation, we implemented support for composable patterns by generating a new event for each fully matched pattern, which is then inserted back into the simulation and can then be used as the basis for other patterns. Although we are dealing with a relatively simple scenario and set of patterns we have covered several important conditions, such as "unless" to terminate this match when invalidated by events, and "unless-forever" to terminate and leave this match in the partial match pool as a block for the pattern ever being matched again. These conditions provide a solid basis for developing more patterns in the future. We also conducted analysis of the performance and operation of the simulation to assess the complexity and stability of our approach.

Our findings demonstrate a stable simulation, with an expected complexity profile, and suggests that our approach does effectively support composable patterns, enabling a more modular approach to the creation of story sifting patterns.

Our work could also act as the foundation for quantitative analysis of future developments such as story sifting intervention – where the system intervenes in order to make partial pattern matches more likely to complete. We have discussed how this could be done with the current simulation for direct and indirect intervention, and how

it might be done with the addition of a protagonist for generative and player intervention.

We are currently conducting quantitative research to evaluate the overall effectiveness of the simulator. In the future, qualitative research may be necessary when our results are applied in the gaming domain, particularly to explore the impact of participant involvement, assess the quality of generated stories, and compare story generation with and without intervention methods.

Emergent narrative has established itself as the basis for a particular form of narrative game with its own aesthetics and its own challenges. Story sifting is one approach to address those challenges without introducing designed elements that might change those aesthetics. We hope that our work might inform the next generation of emergent narrative games, showing how sophisticated stories might be built up from composite patterns, and act as a foundation for future quantitative analysis of how story sifting and story sifting intervention could impact the experience of players.

Acknowledgments

I would like to express my sincere gratitude to my supervisors, Prof. David Millard and Prof. Nicholas Gibbins, for their invaluable guidance and continuous support throughout my studies. Their insightful feedback and encouragement have greatly shaped this research.

I also acknowledge the use of ChatGPT, an AI language model developed by OpenAI, for assisting with grammar corrections and refining the writing in this work.

Lastly, I deeply appreciate the unwavering support of my family, whose encouragement has been a constant source of strength.

References

[1] Ruth Aylett. 1999. Narrative in virtual environments-towards emergent narrative. In *Proceedings of the AAAI Fall Symposium on Narrative Intelligence*. 83–86.

[2] Ruth Aylett. 2000. Emergent narrative, social immersion and "storification". In *Proceedings of the 1st International Workshop on Narrative and Interactive Learning Environments*. 35–44.

[3] Ruth S Aylett, Sandy Louchart, Joao Dias, Ana Paiva, and Marco Vala. 2005. FearNot!–An Experiment in Emergent Narrative. In *Proceedings of the 5th International Working Conference on Intelligent Virtual Agents (IVA2005), Kos, Greece*. Springer, 305–316.

[4] Matt Brown. 2018. Emergent Storytelling Techniques in The Sims. GDC 2018.

[5] Ben Clothier and David E Millard. 2023. Awash: Prospective story sifting intervention for emergent narrative. In *Proceedings of the International Conference on Interactive Digital Storytelling*. Springer, 187–207.

[6] Alain Fournier, Don Fussell, and Loren Carpenter. 1982. Computer rendering of stochastic models. *Commun. ACM* 25, 6 (June 1982), 371–384. https://doi.org/10.1145/358523.358553

[7] Jonas Freiknecht and Wolfgang Effelsberg. 2020. Procedural generation of interactive stories using language models. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*. 1–8.

[8] Tinsley Galyean. 1995. *Narrative Guidance of Interactivity /*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[9] Tinsley A Galyean. 1995. Narrative guidance. In *Proceedings of the AAAI Spring Symposium on Interactive Story Systems: Plot and Character*. 2–729.

[10] Annika Hinze, Kai Sachs, and Alejandro Buchmann. 2009. Event-based applications and enabling technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* (Nashville, Tennessee) *(DEBS '09)*. ACM, 1–15. https://doi.org/10.1145/1619258.1619260

[11] Christian Jacob. 1994. Genetic L-System Programming. In *Parallel Problem Solving from Nature — PPSN III*. Springer, Berlin, Heidelberg, 333–343. https://doi.org/10.1007/3-540-58484-6_277

[12] Shi Johnson-Bey and Michael Mateas. 2021. Centrifuge: A Visual Tool for Authoring Sifting Patterns for Character-Based Simulationist Story Worlds. In *The 17th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment: Programming Languages and Interactive Entertainment Workshop*.

[13] Hartmut Koenitz. 2015. Towards a Specific Theory of Interactive Digital Narrative. In *Interactive Digital Narrative*. Taylor and Francis, 91–105. https://doi.org/10.4324/9781315769189-8

[14] Max Kreminski, Melanie Dickinson, and Michael Mateas. 2021. Winnow: a domain-specific language for incremental story sifting. In *Proceedings of the 17th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 17. 156–163.

[15] Max Kreminski, Melanie Dickinson, and Noah Wardrip-Fruin. 2019. Felt: a simple story sifter. In *Proceedings of the 12th International Conference on Interactive Digital Storytelling (ICIDS 2019), Little Cottonwood Canyon, UT*. Springer, 267–281.

[16] Max Kreminski and Michael Mateas. 2021. A Coauthorship-Centric History of Interactive Emergent Narrative. In *Proceedings of the 14th International Conference on Interactive Digital Storytelling (ICIDS 2021), Tallinn, Estonia*. Springer, 222–235. https://doi.org/10.1007/978-3-030-92300-6_21

[17] Max Kreminski and Michael Mateas. 2021. Toward Narrative Instruments. In *Proceedings of the 14th International Conference on Interactive Digital Storytelling (ICIDS 2021), Tallinn, Estonia*. Springer, 499–508. https://doi.org/10.1007/978-3-030-92300-6_50

[18] Miłosz Markocki. 2021. Reactive Games as an Example of Extensive Use of Evocative Narrative Elements in Digital Games: Cases of Dwarf Fortress and RimWorld. *Studia Humanistyczne AGH* 20, 2 (2021), 71–83.

[19] Stacey Mason, Ceri Stagg, and Noah Wardrip-Fruin. 2019. Lume: a system for procedural story generation. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*. 1–9.

[20] Michael Mateas. 2002. *Interactive Drama, Art and Artificial Intelligence*. Ph. D. Dissertation. Carnegie Mellon University, USA.

[21] Michael Mateas and Andrew Stern. 2003. Integrating plot, character and natural language processing in the interactive drama Façade. In *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE-03)*, Vol. 2.

[22] Michael Mateas and Andrew Stern. 2005. Façade. https://www.playablstudios.com/facade

[23] Maxis. 2003. The Sims 2. Electronic Arts.

[24] Pratama Putra, Jos Tarigan, and Elviawaty Zamzami. 2023. Procedural 2D Dungeon Generation Using Binary Space Partition Algorithm And L-Systems. In *Proceedings of the 2023 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*. 365–369. https://doi.org/10.1109/IC3INA60834.2023.10285811

[25] Mike Rose. 2013. Dwarf Fortress. https://www.bay12games.com/dwarves/ Bay 12 Games.

[26] James Ryan. 2018. *Curating simulated storyworlds*. Ph. D. Dissertation. University of California, Santa Cruz.

[27] Tynan Sylvester. 2018. Rim World. Ludeon Studios.

[28] Milo NR Utsch, Gisele Lobo Pappa, and Luiz Chaimowicz. 2018. Implementation and Analysis of a Non-Deterministic Drama Manager. In *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 167–176.

[29] Richard Walsh. 2011. Emergent Narrative in Interactive Media. *Narrative* 19, 1 (2011), 72–85.

[30] Hong Yu and Mark Riedl. 2013. Data-driven personalized drama management. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 9. 191–197.