Chapter 1

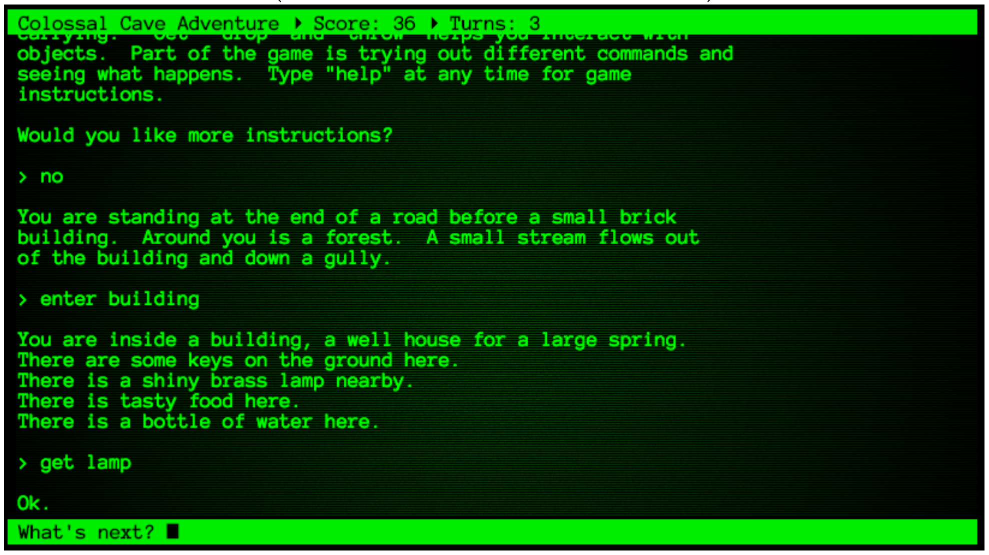
Introduction

The two activities of *playing games* and *telling stories* have shaped the way humans learn and bond with one another since at least the dawn of history. From a humanistic perspective, they are as central to our existence as language [Hui14]. They have also never been entirely distinct, with each drawing on the human ability to invent fictitious worlds, defined by rules and aesthetics originating from people's minds without persistence of truth outside their "magic circles."

Only in a comparatively tiny, recent sliver of humanity's timeline have *digital computers* entered into a place of import for the enhancement of these activities, but they have brought with them an explosion of new thought on how to design and construct deliberate playful, narrative experiences—new magic circles, as it were. One of the central ideas in this rennaissance is that computers enable much richer *system-driven interaction* than we were previously capable of creating. All games, digital or not, form a system, or set of rules that interact with one another nontrivially. For example, the game of Chess has rules for moving pieces of different shapes in patterns that are constrained by rules that depend on the positions of the other pieces. These rules are easy enough for a human to carry out by hand. But thanks to the speed of digital processing, we can now describe highly complex systems whose effects require calculation that would be too tedious to compute by hand while maintaining the spirit of play, such as the trajectories of particles bouncing around in simulated space.

One of the earliest forms of digital game was the *text adventure*, a kind of story told through interaction with a text-based command-line interface to navigate a fictional world. Commands like `go north` and `get lamp` could be typed and translated into changes in the game's state, which would be reflected in changes to the prose describing the player's surroundings, preceding the next prompt for input. (See Figure 1.1 for a sample interaction with one of the earliest known text adventures [CW77].) Because computers came with a means for displaying text already, all of the author's concentration could be left to designing the *narrative world* underneath: things such as the map, the descriptions of places and items, the modeling of reactive elements, such as machines and non-player characters. Items strewn throughout the game world could be collected and combined to solve puzzles, such as fixing broken machines or bribing stubborn characters. Text adventure enthusiasts often speak of the compelling sense of

Figure 1.1: Sample interaction with the game *Adventure* by Will Crowther. (Also known as Colossal Cave.)



```
Colossal Cave Adventure ▸ Score: 36 ▸ Turns: 3
objects.  Part of the game is trying out different commands and
seeing what happens.  Type "help" at any time for game
instructions.

Would you like more instructions?

> no

You are standing at the end of a road before a small brick
building.  Around you is a forest.  A small stream flows out
of the building and down a gully.

> enter building

You are inside a building, a well house for a large spring.
There are some keys on the ground here.
There is a shiny brass lamp nearby.
There is tasty food here.
There is a bottle of water here.

> get lamp

Ok.
What's next? █
```

agency they experience when playing such games: thinking of the right command to type involves a sense of imagination ("What would I do if I were really there?"), which is an entry point into a conversation with the game world in which the player tests the boundaries and rules established by the author. In other words, the player may participate in a *world* created by the game author in order to collaboratively author a *story* with the computer. So the ability to simulate worlds with computers has enriched how we tell stories, too, and has opened up new spaces at the intersection of play and narrative.

After the text adventure's heyday, the mass commercialization of digital games engendered a proliferation of "genre" terms, such as "first-person shooter" and "platformer," that entered the public consciousness as a taxonomy of play. In such a taxonomy, the *story game* is thought of as something entirely separate from the *strategy game*, with the former thought mainly of as alternatives to light reading (interaction mainly "turns the page" in a digital book) whereas the latter is thought of as "more of a game," with real decisions and player agency. These categories then inform how designers conceive of new games, and, at one further layer of indirection, how creators of game design *tools* think to build affordances and libraries. So now it is difficult to discuss, much less create, digital works that focus on things like mechanics, systems, and player agency for narrative play—for example, text games that may require ingenuity and strategy to tell their stories.

A counter-framework for designing and analyzing digital games from first principles was proposed by Hunicke et al. [HLZ04], suggesting a game be thought of in terms of *mechanics*, *dynamics* and *aesthetics* (MDA). Mechanics are the basic rules of the game (Chess example: queens can move diagonally), dynamics are the emergent patterns and strategies that arise from mechanics (Chess example: book openings), and aesthetics are the player's holistic experience of the game (Chess example: the sensory experience created by the pieces and the board; the cognitive experience of thinking several moves ahead to plan a strategy). Such a framework allows one to find similarities in games across artificial "genres" in terms of their mechanics or aesthetics, yet contrast them along another of the three axes.

From a game *creation* perspective, the MDA framework also enables us to imagine building a model of the mechanics separately from the game's rendering as a whole, aesthetic experience. This approach lends itself well to *rapid prototyping*, the idea that designers do not want to build a complete end-to-end product before iterating on their initial design, and so they should build rough but servicable mock-ups (prototypes) until the design seems likely to work. Especially when creating games based on complex systems with lots of interacting components, it can be helpful to test those systems piece-by-piece before they are too large to reason about.

From a computer science perspective, we are in search of *composable abstractions* for understanding narratives and games. The process of designing those abstractions, together with defining how they may be brought to life through execution and interaction, is what constitutes the design of a programming language. The executable model should give the designer some sense for the dynamics (and perhaps aesthetics) of the eventual finished game, and it may also provide some automation to speed development: automatically testing the game by carrying out some designated player strategies, for example. It could also perform analysis of the program and tell the author whether or not it has met some desired game properties. Such analysis plays a similar role to *regression testing* in current-day software practice, but it has the ability to check exhaustively for adherence to the specification.

Creating new abstractions for understanding different classes of computation has been the purview of the field of programming languages for several decades. Before digital computers had been invented, we had notions of computation that were independent of the particular physical artifacts for carrying it out, such as Church's λ-calculus [Chu32]. After a long detour through the "von Neumann bottleneck" [Bac78], we can now write programs in these machine-independent language models and run them on real machines. Meanwhile, game designers have been inventing their own programming languages, such as Inform 7 [Nel01], Twine [KA08], and Versu [ES], that suit the specific needs of people crafting interactive stories with incredible immediacy, in terms of affordances that match their imaginative process. But too often, these languages lean on the tradition of machine dependence (imperative state update) to establish their meaning, resulting in leaky or unstable abstractions. We champion the goal of these tools— accessible means for non-computer scientists to participate in the creation of games—but suggest that an approach based on machine-independent, linguistic techniques would serve the community. Furthermore, in sight of the broader goal of *understanding* games at a basic and timeless level, we posit that neither the class of games in question nor the programming tools in question should depend on the trends nor limitations of current technology.

As such: this thesis project endeavors to provide a conceptual and computational framework for understanding, building, and analyzing the interactive worlds that underlie games and stories.

1.1 Potential Narratives

Text adventures eventually came to be understood under the larger umbrella term *interactive fiction*, or interactive works that tell stories with words. Like computation itself, the history of interactive fiction begins before (and continues to exist outside of) the digital computer.

Nick Montfort, an interactive fiction author and historian, traces the history of interactive fiction to the *I Ching*, an ancient Chinese divination tool that used random chance to generate prophetic text, and to the Llull machine, a similar Western invention [Mon05]. The connection is that these systems generate text with algorithms, based on some underlying model, much like an interaction with a digital computer to derive a story. In the 1960s, the French surrealist group of mathematicians, artists, and other intellectuals going under the name Ouvroir de Littérature Potentielle, or *Oulipo* [Mot86], gave the term *potential literature* to these systems that were not stories themselves, but produced stories through human interaction. Following in step, we adopt Montfort's term *potential narrative* to describe our approach to narrative play, because we find it a useful way to unify the playful aspects of interactive fiction with more readily-accepted notions of game.

Interleaved with the development of digital interactive fiction were the "Choose Your Own Adventure" and "game book" collections of print media devised as youth entertainment, which present passages of text ending in choices, each of which corresponds to a page in the book that the reader should turn to to continue the story. [HWF10] These stories with enumerated choices at each branching point bear a strong structural similarity to the digital form of *hypertext*, now widely known as the format of the World Wide Web: text is organized into passages, each of which can contain *links* to other passages. A resurgence of interest in hypertext narratives came with the introduction of Twine [KA08], an accessible tool for creation in this form. Even these more finitary notions of participatory storytelling can be thought of as potential narratives: each path through the story graph corresponds to one story.

Even as text-based digital games have fallen out of mainstream fashion in favor of the dazzling possibilities afforded by modern graphics cards, storytelling still drives the imagination behind widely-acclaimed commercial games, for everything from the grand-scale adventure of Skyrim [Stu11] to the more personal world of Gone Home [Gay13]. The idea of inhabiting a fictional persona to explore and change a rich, reactive environment has seemingly never fallen out of favor, and the new experiences enabled by digital games seem to have the potential for lasting impact on human culture as much as literary and cinematic fiction is now accepted to have. By tapping into both historical precedent and computational possibilities, understanding these timeless forms of art can bridge the humanities and sciences, fostering new means of expanding human knowledge.

The relevance of interactive fiction to the broader domain of games and interactive simulations that I investigate in this dissertation is less about the fact that they tell stories with words and more about the fact that they *free the game designer* from certain constraints that are present when thinking in terms of graphical motion rendering. Emily

Short and Richard Evans, the authors of the interactive storytelling authoring system Versu, write about a similar concern in terms of the *composability* of text versus other generative systems, and while we are more concerned with internal representation than rendering, their explanation illustrates the relative cognitive concerns involved. [ES]

The main abstraction that our work borrows from interactive fiction and storytelling is that of describing a *possible event space* in which an actor may intervene. As a domain of computer science, interactive fiction contains interesting problems in the systems that underlie it, and the questions of how to make those systems more expressive.

1.2 Generative Systems

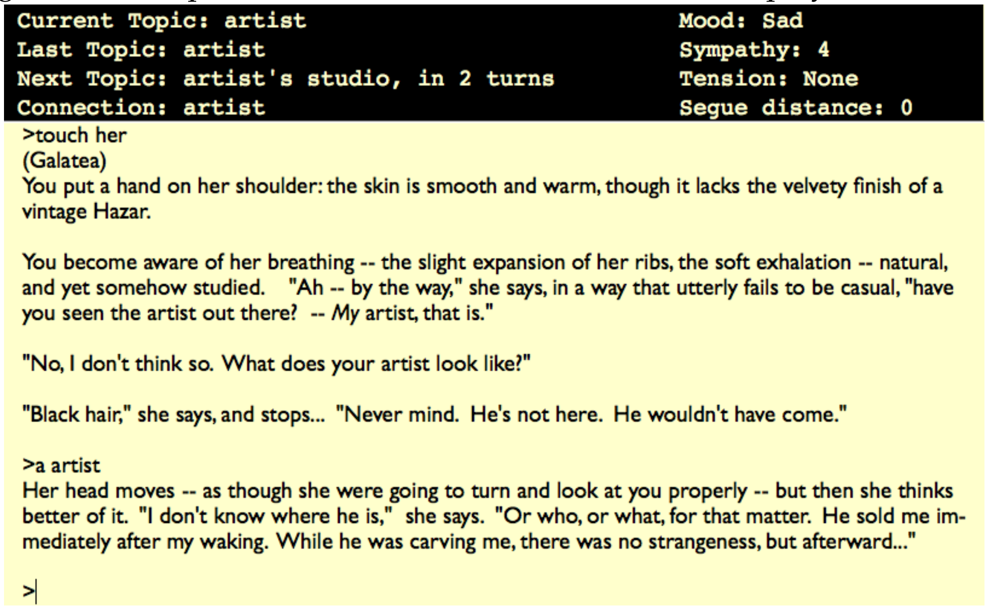
One of the central problems in expressive interactive narrative design also concerns many other artistic and scientific disciplines. The problem is one of human-created versus computer-created content. For any piece of media that changes in response to interactor input, the changes need to be generated on-demand. This means that any sequence of interactions needs to create a representable state. In the case of branching stories, any sequence of *choices* should result in a meaningful narrative situation. One possibility, and the one chosen for Choose Your Own Adventure books and hypertext, is for an author to create the amount of content necessary for every possible run in the world simulation. But when a world is composed of many components, such as a player inventory, the problem becomes intractable: every new variable doubles the number of possible states. Instead, computational methods can be used to create new scenes on-the-fly without having been seeded with them by the author. For instance, a data structure to track the player inventory across different rooms can be seen as a tool for generating a large number of story states. We refer to the techniques involved in systematically composing a small number of hand-generated pieces to derive a large state space as *generative methods*, though it is sometimes referred to as *procedural content generation* [COM13]. We review a few examples of games using generative methods that we take as inspiration for the kind of system we are interested in finding abstractions to describe.

One domain for such methods is the simulation of non-player characters (NPCs). A game can track a detailed model of character *interiority* [Pot05] including mood, knowledge, sentiment, and goals. *Galatea*, an interactive fiction game by Emily Short [Sho00], uses a complex dialogue model to create a "living statue" character that reacts differently to player commands depending on her mood, sympathy, tension, and conversation history and topics (see Figure 1.2). The system generates a wide space of possible stories due to variations with respect to these factors. Short implemented Galatea using a quip-and-response dialogue model reminiscent of a state machine. [1]

Another problem domain is the simulation of multiple characters or agents. Will Wright is an acclaimed game designer known for systems that generate worlds. He is the primary designer behind the Sims series, including SimCity [Wri96] and The

[1]See `https://emshort.wordpress.com/how-to-play/writing-if/my-articles/conversation/` for Short's own explanation.

Figure 1.2: Sample interaction with *Galatea* with state display tool enabled.



```
Current Topic: artist                        Mood: Sad
Last Topic: artist                           Sympathy: 4
Next Topic: artist's studio, in 2 turns      Tension: None
Connection: artist                           Segue distance: 0
```

>touch her
(Galatea)
You put a hand on her shoulder: the skin is smooth and warm, though it lacks the velvety finish of a vintage Hazar.

You become aware of her breathing -- the slight expansion of her ribs, the soft exhalation -- natural, and yet somehow studied. "Ah -- by the way," she says, in a way that utterly fails to be casual, "have you seen the artist out there? -- *My* artist, that is."

"No, I don't think so. What does your artist look like?"

"Black hair," she says, and stops... "Never mind. He's not here. He wouldn't have come."

>a artist
Her head moves -- as though she were going to turn and look at you properly -- but then she thinks better of it. "I don't know where he is," she says. "Or who, or what, for that matter. He sold me immediately after my waking. While he was carving me, there was no strangeness, but afterward..."

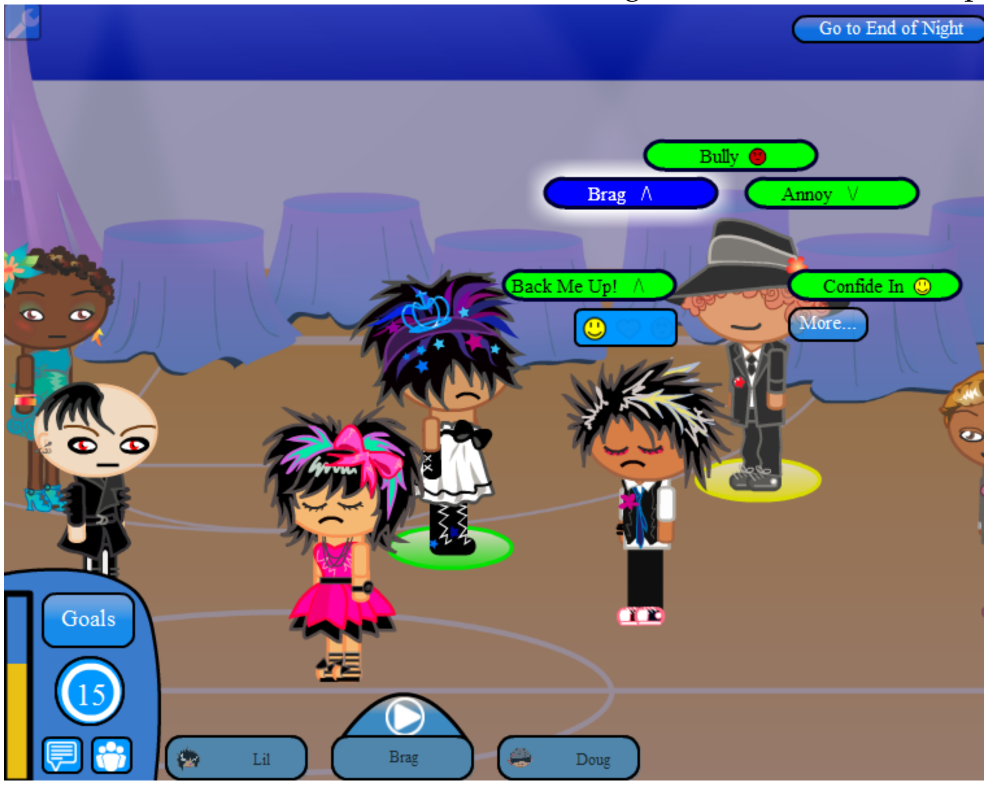
>|

Sims [Wri00]. None of the systems in these games are explicitly stated as *potential narratives*, but they nonetheless allow human collaboration to tell stories. The connection with The Sims is perhaps most clear: the game provides a "digital dollhouse" with which to perform stories at the player's whim, without imposing external narrative goals. Sim City works similarly, but with a focus on city configuration and aggregate population data rather than the personal relationships and goals of the Sims. Spore [Art08] was a virtuosic attempt to include simulations at multiple levels within an overarching meta-system. All of these systems compose experiences out of myriad changing state components, requiring very little hand-authored content. The idea that a story can nonetheless be derived from such a system is sometimes called *emergent narrative* [Ayl99]. The implementation challenges of such a system, as far as we are aware, mainly lie in managing a very large evolving state (i.e. the status elements of every individual Sim).

More recent investigations into the space of generative narrative experiences include the high school social simulation game, Prom Week, which uses sophisticated social models to determine how characters react to actions such as bullying or confiding based on their own sentiments toward and memories of the acting character (see Figure 1.3). Prom Week grew out of a tradition in interactive drama that includes the landmark production of Façade [MS03], i.e. "believable drama." Their aim was to create a system that would generate a story using *social physics*, or rules for character interaction, rather than presenting hand-authored scenes. The primary technology driving Prom Week is actually a language, Comme il Faut (CiF) [MTS$^+$10], in which the social physics engine is implemented. Inference based on character interiority is used to generate possible intents, which combine with player selection to determine actions.

We present these examples as a (partial) survey of state-of-the-art techniques in

Figure 1.3: Screenshot of *Prom Week* demonstrating character interaction options.



playable narrative based on generative methods. Having established this context, we can now describe the goals, methods, and contributions of the dissertation.

1.3 Thesis Goals

Our overarching goal, extending beyond the scope of this thesis but informing its approach, is to develop computational tools and conceptual frameworks that support creating novel, experimental games and narrative systems. We frame the problem as one of *programming language design* in the sense that this perspective entails creating new abstractions with platform-independent, mathematical semantics. We also aim to bring high-level design language in closer alignment with programming constructs, i.e. make it so that the means of expressing ideas as executable artifacts closely mirrors game designers' conceptual framing.

The narrower goal for this thesis is to fully develop a correspondence between a theory of formal logic proofs and ideas in narrative and game design, and to realize its consequences in the form of a modeling tool. We have noticed a strong match between recently-emerging programming methodologies—specifically *forward-chaining, substructural logic programming*—and the idioms needed to express game worlds and mechanics, especially interactive fiction, and by exploring this connection we aim to advance ideas in both fields: new programming languages for game design can expand designer creativity potential, and new application domains for programming languages can expand the range of computing phenomena accounted for by mathematical understanding.

Our target audience for this work is mainly research communities within game design and logic, although we aspire to create an appeal to (non-academic) systems-oriented narrative and game designers, and programmers with an amateur interest in game design. Ideally, this work will serve to bridge communities interested in programming language theory and game design and reveal many of the overlapping concerns: for instance, both programming language design and game design consitute the creation of *interfaces* with a human user.

As artifacts whose goals are often dependent on human cognition such as *expressiveness*, *productivity*, *flow*, and *emotional impact*, both games and programming languages are difficult to evaluate in quantitative measures, at least with purely computer science-based techniques. If we intend to deepen a holistic understanding of activities like play and programming, clearly some humanistic and psychological understanding will be needed. Nonetheless, there are still questions that may be answered about the computer's half of the bargain, and indeed that can be answered with precise mathematics. It is those questions that we hope to bring into greater clarity for games through methodologies commonly practiced in programming language theory.

In programming languages, the search for objective evaluation criteria has turned mainly to mathematical proof: we create a formal model of a new programming language design, then prove its soundness with respect to the appropriate formal notions. For instance, a common theorem proved for a new language is *type safety*, which says that certain rules circumscribing the set of admissible programs in the language ensure that no program in that set will crash nor reach a stuck point in its evaluation. One can imagine by analogy that the rules circumscribing a *game* might also ensure some formal properties: for instance, in a multiplayer board game, we might want to ensure that play is *productive* in the sense that as long as the game is not in an end state, some player can always make a move that changes the game. The notion corresponding to a *program* in this setting is a *playthrough* of the game.

But before these proofs can be formalized, and thus before we can truly subject games to mathematical evaluation, we need a language for codifying playful rule systems themselves. This thesis describes a candidate for such a language, which is based on a logical system called *linear logic*. We demonstrate how narrative structure can be described abstractly in linear logic, then go on to see how we can realize linear logic as a suitable programming language.

Part of this thesis is a new programming language designed specifically for the support of narrative and game design, and its own design has the following goals:

1. Has a mathematical foundation for the sake of clear, portable, and extensible semantics
2. Is general enough to describe a wide range of operational logics in games
3. Can describe important aspect of narrative structure, supporting causal reasoning and dependency analysis between narrative events

4. Has the potential for accessible front-end tools to be built atop it.

Our evaluation methods are twofold: first, by a selection of case studies using the language, and second, by mathematical proofs about the language.

For the first criterion, we develop case studies that we believe to be representative of the generative storytelling ideas surveyed in this introduction. The code for each of these studies, as well as our reports on the iterative development process, serve as the non-quantitative results of the thesis, available for subjective evaluation to anyone who might consider using the language. Our own subjective judgment on the success of these case studies is presented at the end of Chapter 5.

For the second, we illustrate the use of a mathematical reasoning framework for stating and proving theorems about games encoded in our system. We prove that, in general, questions of whether certain properties hold of games are decidable. Thus, automated checking of designer-stated properties of games may potentially be integrated into the game prototyping and development process.

1.4 Approach

Now that we have stated our goals for the thesis, we may be more specific about the computer science tools that we use to achieve them.

1.4.1 Preliminary Definitions

While play and stories as aspects of human culture elude fully-characterizing definitions, we nonetheless need to define the scope of this thesis. Below we provide some working definitions.

We define a *world* as a *state space* together with *mechanics*. A *state space* is a set of parameters that can take on values that evolve over time. For example, the *state space* of Chess includes the board, pieces, the locations and captured status of the pieces, and which player's turn is in effect. A *state* is an element of the state space, e.g. a particular configuration of pieces on a Chess board. Mechanics are rules for how states can change.

Worlds may also specify one or several permissible *initial states*, in which case they can be *run*, or evolved step by step according to its mechanics.

An *interactive world* is a world whose mechanics allow human participation or intervention while the game is running. We can supplement this definition with informal, subjective notions to say when an interactive world is a *game* or a *potential narrative*; that is, an interactive world is a game when the interactor feels that they are *playing* it, and an interactive world contains *narrative* (or is a *story world*) when its runs can be understood by a human as stories.

The one aspect of these definitions that has not been completely detailed is what is meant by *rules* for state change. By a rule, we mean something approximately of the form "*If X then Y*," often codified as logical implication $X \supset Y$ ("X implies Y"). But standard logical implication does not adequately capture the idea of transforming state, as we explain next, leading us to an alternative formalism.

1.4.2 Linear Logic

Game rules refer not to a monotonically increasing body of knowledge but to a heterogeneously evolving state. They tell us what is permissible, but also what changes when an action is carried out. The fact that a player's rook is in a given Chess board cell may hold at one point in the game, but after an action is taken, its position may change. So it is not a "fact" in the same way that $1 + 1 = 2$ is a fact; whether it holds depends on *time* or some abstraction of it.

Logics of action and time have been studied well beyond the specific domain of games, and include temporal logic [Lam94], situation calculus [MH69], and event calculus [KS89]. All of these calculi stratify logical propositions by explicit timesteps, and action specifications increment the timestep. However, in order to model a world with a complex state space containing many components, each of these logics needs a notion of *inertia* or *frame rule* that says "whenever an action changes the state of some components, the rest of the state stays the same." These rules need to be axiomatized, leading to unwieldy proofs. Additionally, most applications and extensions of these logics have been concerned with adapting them to model the real world (e.g. for robotics applications) such that they do not easily suit the world of narrative fictions [RY10].

Jean-Yves Girard's *linear logic* [Gir87] is a logic that captures essential notions of state change without explicit reference to time. The observation behind linear logic is that standard logics' monotonicity could be accounted for by the so-called *structural rules*, or rules defining how assumed or known facts could be used in a proof—specifically, assumptions may be freely duplicated and ignored. If one explicitly marks the duplication and ignoring of assumptions, rather than building them into the logic, then what emerges is a notion of logical consequence embodying *state transition*, or evolving systems at a component-wise level, rather than permanent knowledge. Furthermore, linear logical deductions are structured in such a way as to track *resource dependency* between different state transitions, making them subject to analysis in terms of causality.

Thus, linear logic's nature as a proof-theoretically-justified "logic of changes" makes it suitable for our purposes to desribe narrative actions and game rules in this thesis. We detail its definition and use for narrative spefications in Chapter 2.

1.4.3 Logic Programming

Logic programming is the main technique we will use to link the logical with the computational. Traditionally, logic programming is realized by languages such as Prolog,[2] wherein programs are collections of clauses (logical implications) together with a *query*, or directive for proof search. A proof search engine then executes the program by attempting to assemble a proof of the query from the program clauses. For example, a program defining addition as a logical predicate `plus(X,Y,Z)` that holds whenever `X+Y=Z` may be given the query `plus(X,Y,10)`, which would compute all pairs of numbers that sum to 10. In general, there may be many proofs or no proofs. Generally one

[2]See `http://www.swi-prolog.org/` for one implementation.

is interested primarily in *whether or not* there is a proof, and if so, which terms satisfy it. In this domain, the engine searches for a proof with *backward chaining*, or reasoning backwards from a goal to the assumptions available.

However, in our case, the *proof itself* is a relevant artifact of study—it embodies the causal dependencies between events in the interactive world, which we illustrate in Chapter 3. In the case of simulation and generation of narratives, as well as open world games, we are often not especially interested in a particular goal (i.e. outcome of the simulation) so much as the process and intermediate states, which are recorded by the proof itself. In particular, we construct proofs with *forward chaining*, or reasoning forward from a set of assumptions (representing an initial world configuration), to model evolution of the interactive world "forward in time."

Because of our shift in focus from goal-based inference to forward simulation, we distinguish *proof search*—a process that may backtrack, and typically performs an exhaustive search of the provability space—from *proof construction*, or the use of logically valid deductions to navigate a space while not necessarily arriving at a particular goal. Put another way, proof construction may sacrifice *completeness* of provability, but not soundness.

In addition to its primary function in our domain as a representation system for rewriting rules over partial states, logic programming offers support for a variety of powerful computational idioms, such as user-defined datatypes, relation-based definitions, and pattern matching on narrative states. Many of these idioms are also found in approaches developed by the Artificial Intelligence (AI) community, such as planning and generative grammars. Many researchers in games have applied similar techniques and referred to them as "AI techniques" [TZE$^+$15] rather than "programming language features"—by presenting these ideas from a programming language point of view, we hope to bring some unification to ideas across these fields.

1.4.4 Logical Frameworks

A final source of techniques that we use as a starting point for our work is *logical frameworks* as described by Harper et al. [HHP93]. The aim of logical frameworks is to provide a basis for formal codification of deductive systems in such a way that theorems can be stated and proved by a human, and checked by computer, within that system. The logical framework LF was realized as the proof assistant Twelf [PS98] for this activity. Twelf's underlying machinery is largely based on logic programming, although the "logic programs" written within it are typically analyzed, not executed. Logical frameworks research has experimented with extending the underlying logic of Twelf to linear logic, and the community has since been searching for suitable notions of specification such that similar meta-theoretic capabilities to Twelf are possible. We rely heavily on progress in this area, and contribute to that progress, for our concerns around proving properties of games in Chapter 6.

1.5 Contributions

This dissertation describes the use of linear logic to specify potential narratives and game mechanics. Our primary contribution is a full explication of a deep correspondence between *proof* and *play*, or alternatively *proof* and *narrative*, in a way that gives a formal justification for unifying play and narrative both conceptually and for the sake of computational tools that can aid their development.

The formal model is implemented as two different logic programming languages, one existing prior to this thesis work (Celf) and one implemented for the purposes of deeper investigation into this work (Ceptre). Essentially, these investigations serve the purpose of pushing the envelope of the formalism as far as it can go to find the limits of its suitability for, and correspondence with, games and potential narrative specifications.

Our specific claim follows in the form of a thesis statement.

> **Thesis statement:** *Using linear logic to model interactive worlds enables rapid prototyping of experimental game designs and deeper understanding of narrative structure.*

As evidence for this claim, we provide a written explanation of the correspondence between linear logic proofs and stories, then illustrate the use of that correspondence for *narrative generation* (in Celf) and *game prototyping* (in Ceptre). These results extend the limits of present knowledge in how formal logic can be used to support generative methods and interactive world authoring, and they also provide supporting case studies for a particular methodology in programing language design. We detail how each part of the dissertation aligns with the thesis statement below.

Chapter 2 describes the correspondence between linear logic proofs and causally-structured story, building largely on work by Bosser et al. [BCC10]. This chapter mainly serves to establish the conceptual framework in which we understand narrative and simulation events, separate from the notions of generation and interaction.

Chapter 3 shows how to *operationalize* the interpretation of narratives in linear logic as programs, via a standard and general extrapolation of logical definition into proof construction procedure, known as logic programming. In this chapter we illustrate how to use the programming language Celf to generate narratives through a narrative world specification given in terms of linear logic. This chapter supports the *deeper understanding of narrative structure* part of the thesis statement.

Chapter 4 describes the development of a new programming language, Ceptre, suitable for programming not just *generative* but also *interactive* worlds. We introduce a new programming construct called *stages* that serve to partition a world specification into parts, then coordinate the control flow between those parts, resulting in a practical tool for prototyping game mechanics and simulations.

Chapter 5 presents several case studies of Ceptre's use for developing interesting examples in the union of narrative and play design. We posit this chapter as the primary support for the *rapid prototyping* part of the thesis statement.

In Chapter 6, we describe progress toward reasoning tools for linear logic programs that allow authors to specify and check constraints on the evolving state. Such tools, once fully automated, would allow world authors to specify their intent and ensure that program rules preserve that intent. This chapter serves as secondary support for the *rapid prototyping* part of the thesis statement, since program errors can often cause slow progress toward building a working prototype.

Finally, in Chapter 7, we conclude by recapitulating the thesis statement, summarizing how the work above has fulfilled its promises, and suggesting avenues of future work.