Moirai: Enabling Complex Narrative Structure in Simulation-Driven Stories

Ben Samuel[1], Adam Summerville[2]

[1]University of New Orleans, 2000 Lakeshore Drive, New Orleans, LA, 70148
[2]The Molasses Flood, 51 Sawyer Road, Waltham MA, 02453

Abstract

Moirai is a system that governs a social simulation experience from initialization, through the simulation, to its end. Previous social simulation experiences have usually utilized bespoke glue code that chains together the different aspects of running a social simulation, utilizing simulation either before play, during play, or after play. *Moirai* is capable of chaining together different initialization passes and simulation modules to enable a range of experiences that have not been shown before. This paper presents the syntax and semantics of the *Moirai* Domain Specific Language, and demonstrates how it can be used to create a variety of complex simulation-driven narratives inspired by case-studies from previously existing media.

Keywords

social simulation, procedural narrative, domain specific language

1. Introduction

In Greek mythology the *Moirai* were the the sisters of fate – Clotho, Lachesis, Atropos. In this paper, we discuss *Moirai* a system that governs a social simulation experience from initialization (*Clotho* or "spinner" the one who presided over birth), through the simulation (*Lachesis* or "allotter" the one who controlled the goings on of life), to stopping simulation (*Atropos* or "unturnable" the one who severed the string of life). *Moirai* uses a Domain Specific Language (DSL) that enables a user to set up a social simulation experience, covering a wide range of possible experiences.

Previous social simulation experiences have usually utilized bespoke glue code that chains together the different aspects of running a social simulation. These experiences have utilized social simulation at different times of the experience:

- **Before** Play – to set up the experience
- **During** Play – to provide the simulation *as* the experience
- **After** Play – to provide an epilogue

with all previous experiences (to our knowledge) using at most two of these aspects. Furthermore, these experiences have generally progressed in this one direction, with no branching or looping, limiting the types of experience possible.

Moirai is capable of chaining together different initialization passes, simulation modules, and filtering of population which enables a range of experiences that have not been shown before. Of course, to not overstate our claim, the previous experiences have been authored (at some level) with Turing complete languages, so there is nothing inherently possible with *Moirai* that could not also be achieved in those languages. However, by providing a DSL that elevates certain operations the authoring experience is greatly simplified.

2. Related Work

Moirai is a system intended to be used in conjunction with social simulations. By social simulation systems, we refer to computer processes that simulate individual entities (often referred to as non-player characters, or NPCs [1]), social relationships between them, and frequently also has a hand in shaping the world these entities inhabit. Social simulations are often not intended to be stand-alone, but rather integrate with and enable interactive experiences. For example, the social simulation system *Talk of the Town* [2] is responsible for generating the residents of the fictional small American town in which the performance art piece *Bad News* [3] is set. Likewise, the social simulation engines *Comme il Faut* and *Ensemble* [4, 5] have been core to experiences such as *Prom Week*, *Vox Populi*, *VESPACE* [6, 7] and enabled mods for AAA games such as *Skyrim* and *Conan Exiles* [8, 9]. While not yet used for any experiences, *Kismet* [10] is a social simulation language designed under the notion of Compton's "Casual Creators" [11] that aims to provide pleasing expressive range [12] with a relatively low complexity barrier to entry.

The work outlined in this paper explores the use of simulation at different parts of an experience. To this end, under a broader view of "social simulation" as any

AIIDE Workshop on Experimental Artificial Intelligence in Games, October 08, 2023, University of Utah, Utah, USA
bsamuel@cs.uno.edu (B. Samuel); adamsumm@gmail.com (A. Summerville)

rule driven process during which characters grow and dissolve different relationships, the authors take inspiration from a number of experiences that have utilized varying levels of simulation. *Friends at the Table* – an actual play podcast that uses games as the substrate to tell an audio play – have used multiple games as setup for longer arcs. In the simplest examples, for their *Marielda* arc [13], they played *The Quiet Year* [14] – a game where players take turns filling out a map – to develop the world for a longer run of *Blades in the Dark* [15] – a more standard table top role playing game. *The Adventure Zone* [16] similarly used *The Quiet Year* to develop the world for their *Ethersea* arc [17] and then used *Dungeons & Dragons* [18] for the more standard segment. *Friends at the Table* has also gone even more in depth during their *Partizan* [19] arc which used seven different games for worldbuilding at various levels of fidelity – starting with *Dialect: A Game About Language and How It Dies* [20] to first develop the language and culture of their game's world, then moving into *Armour Astir: Advent* [21] to develop a conflict, *Ech0* [22] and *Dusk to Midnight* [23] to explore the aftermath of that conflict, *Beam Saber* [24] and *For the Queen* [25] to continue on with the conflict, and finally *Microscope* [26] to play out the 500 years after the end of the conflict.

3. *Moirai* – Method & Capabilities

In the following section we will discuss the syntax and semantics of *Moirai*. At its core, *Moirai* is an imperative scripting language with standard control flow and branching operations. However, unlike other scripting languages, *Moirai* provides a range of language constructs for initializing, running, swapping, and terminating social simulations. Currently, *Moirai* is designed to be used with *Clotho* – a social simulation initialization system– and *Lachesis* – a social simulation engine that the authors have developed (previously known as *Kismet* [10]). However, it would be possible to use other creation and simulation engines, so long as they follow the APIs of *Clotho* and *Lachesis* with abilities to be initialized, queried, ran, and serialized.

The specifics of *Moirai* can be broken down into four main properties: (1) the loading/unloading of ruleset files (i.e., files which dictate what is able to transpire within the simulation), (2) the creation and filtering of simulated entities (i.e., the characters, locations, and other elements driven by the simulation), (3) the running of simulations (i.e., having the simulated entities change based on the possibilities specified by the rulesets), and (4) the control flow governing these operations (i.e., repetition and selection of the aforementioned affordances). These four properties will now be discussed in further detail.

3.1. Loading/Unloading

One aspect of social simulations that *Moirai* operationalizes is the use of different rulesets. Existing simulations often use different rulesets at various points of simulation. For instance, the rules that govern the history generation in the simulation-heavy *Dwarf Fortress* [27] are different from those that govern the moment-to-moment interactions of dwarves. Similarly, *Bad News* had simulations that operated at different levels of fidelity with the bulk of simulated time happening at a very coarse level of simulation with only the final simulated week (of a simulated 100+ years) happening at a granular level.

Toward this end, *Moirai* provides the functionality to load in, add, and remove different rulesets, while keeping the simulated entities in place. By rulesets, we refer to specifications that dictate what behaviors simulated entities can engage in, any preconditions those behaviors may have, postconditions for how simulated entities might be affected by taking the behavior, etc. Furthermore, inspired by how Tabletop Role-Playing Games (TTRPG) communities build off of existing rulesets with new modules, it is possible to load in multiple ruleset modules at once. For instance, if there existed a generic high school simulation module, it could be mixed with a supernatural horror module to create mixed-genre simulations in the vein of *Buffy the Vampire Slayer* [28] or *Stranger Things* [29]. Alternatively, swap out supernatural horror with a romance module to create high school romance simulation games like *Tokimeki Memorial* [30] or *Katawa Shouju* [31].

The operations available in *Moirai* are:

- `load RULESET+`
  unloads any currently loaded simulation rulesets and loads the given simulation ruleset files. The "+" indicates "one or more" (and retains this meaning throughout the rest of our descriptions of *Moirai*'s operations)
- `add RULESET+`
  loads the given simulation ruleset files and concatenates them to the already loaded rulesets
- `remove RULESET+`
  unloads the given ruleset files and removes them from the current rulesets

All of the operations leave the simulated entities in place throughout the adding, removing, and swapping of rulesets. The only major consideration to this is that the simulated entities should have consistent attributes and hooks across the modules. For instance, if our "high school" ruleset treated "degree of talkativeness" as a scalar variable referred to as *extroversion*, and the "supernatural horror" ruleset treated it as a ternary value of *introvert/normal/extrovert*, then the two modules wouldn't

realize these two variables are referring to to the same notion, leading to the modules failing to fully leverage their full potential synergy.

In the next section we will discuss *Moirai*'s operations for handling entity creation, filtering, and removal.

3.2. Simulated Entity Creation and Management

All social simulation based experiences share the commonality that there are entities that are being simulated. Although in theory anything can be a simulated entity, to provide the reader a frame of reference simulated entities are often locations (e.g., a dining room in a home), and the creatures and things that populate them (e.g., a dining table, a person working on it, a cat napping beneath it, etc.). These entities might have been authored to provide a known starting point (e.g., *Prom Week*, *Princess Maker* [32], etc.) or procedurally generated (e.g., *Dwarf Fortress*, *Bad News*) or possibly a mixture of both (e.g., *Crusader Kings III* [33]). Most commonly, once the entities are initialized and social simulation occurs they will continue on according to the rules of the simulation, only leaving the simulation when the simulation itself decides so (e.g., death or leaving town in *The Sims* [34] or *Bad News*). However, many common story patterns found in literature, film, and interactive experiences involve casts of characters coming in and out of focus depending on the narrative. For instance, in a closed room murder such as *Murder on the Orient Express* [35] or *Gosford Park* [36] a subset of characters is involved in the actual murder mystery while other characters might be involved in their – possibly linked – back stories.

Toward enabling social simulation modalities that explore new use cases, *Moirai* enables entities to be initialized, filtered, stored, and restored. In conjunction with the ability to swap simulation rulesets, this allows for different story patterns to be possible, as opposed to just changing the rulesets alone. To the authors' knowledge, the only existing social simulation experience to do anything of this form is *Dwarf Fortress* which creates a new dwarf expedition at the start of Fortress mode.

Entity creation and management in handled with a single operation "initialize", though its execution can be customized through eight options. The syntax and explanation of these operations and options in *Moirai* are:

- `initialize INITIALIZATION+ [:OPTION]+`
  initializes entities according to the INITIALIZATION files (which may be either deterministically authored or procedurally generated) with OPTIONs on what to do with the existing entities (if any)
- **OPTION:**
  `restoring entities in VARIABLE`
  restores entities that were previously stored in the variable named VARIABLE
- **OPTION:**
  `keeping all (characters|locations)`
  keeps all entities of the given class. The | symbol in this context represents an OR selection between the symbols in the parentheses (and does in all subsequent operation explanations).
- **OPTION:**
  `keeping entities where is STATUS`
  keeps entities that have the STATUS property
- **OPTION:**
  `keeping entities where in PATTERN/ARITY`
  keeps entities that have are in the PATTERN that has arity of ARITY. This is to disambiguate between patterns with the same name but different #'s of entities (e.g., the patterns lovers/2 and lovers/3 referring to couples and thruples, respectively)
- **OPTION:**
  ```
  keeping entities where (in
  ↪ PATTERN1/ARITY1 | RELATIONSHIP)
  ↪ with entity (is STATUS |  in
  ↪ PATTERN2/ARITY2)
  ```
  keeps entities that have the RELATIONSHIP or are in PATTERN1 with an entity that either has STATUS or is in the PATTERN2
- **OPTION:**
  ```
  keeping entities where  (in
  ↪ PATTERN/ARITY | RELATIONSHIP) with
  ↪ entity in kept
  ```
  keeps entities that have the RELATIONSHIP or are in PATTERN with an entity that has already been kept
- **OPTION:**
  `stashing rest in STASH`
  saves the entities that have **not** been kept in the variable STASH
- **OPTION:**
  `saving kept as STASH`
  saves the entities that **have** been kept in the variable STASH

Moirai assumes that the given simulation engine has relationships which connect simulated entities (most likely a given, since a social simulation without relationships would be relatively boring) and the ability to define arbitrary patterns (e.g., the *love triangle* pattern between three characters labelled RivalA, RivalB, and Target where RivalA and RivalB both love Target, or the *unrequited love* pattern between two characters, Source and Target, where Source loves Target, but Target does not love Source, etc.). These patterns can be viewed through the lens of *story sifting* [37] as larger patterns that emerge from the simulation, but which might not

be directly represented in the simulation ruleset files themselves.

The options to filter can be at the unary level (an entity has a given attribute), the binary level (an entity is in a given relationship with another entity), or at the *n*-ary level (an entity is found in a given pattern with other entities), with the higher levels allowing to match the other entities against other statuses or patterns or just looking at entities that are already kept.

To give a concrete example of why this might be useful, let's imagine a simulation focused on a wedding (such as *Father of the Bride* [38]). For this example, imagine the wedding as a secondary simulation as part of a larger simulation. That is, the larger simulation has already created, let's say, a town full of people with varied relationship connections across them, and within this town are two people primed to get married. To accommodate their simulated nuptials, first, we would need to initialize the wedding-specific facets of the simulation (setting up the venue, perhaps instantiating the officiant, planner, florist, etc.), by specifying an initialization file

```
initialize wedding:
```

The following are all options concatenated to this initialization. After setting up the wedding, the next order of business is to find the couple in the simulation who is getting married:

```
keeping entities in pattern getting_married/2
```

then we would want to invite the families of the (possibly) happy couple (or, perhaps more accurately, *not filter them out* of the simulation):

```
keeping entities related_to entities in
 ↪ getting_married/2
```

we'll invite/keep the couples' friends, too:

```
keeping entities in pattern
 ↪ best_friend_of_couple/3 with entities in
 ↪ getting_married/2
```

and then we will save everyone else that was simulated in a separate variable, so that they can be returned to after the wedding specific simulation without being included in the wedding simulation:

```
stashing rest in not_attending_wedding
```

After initialization (or perhaps re-initialization), the actual simulation of the wedding itself needs to play out, which we will discuss in the next section.

3.3. Simulation Management

Moirai assumes that the given simulation engine can take one step at a time, and in between steps, the engine can be queried to find not only if a given set of conditions is true, but also the counts of how many entities for which the conditions are true. There exists a single operation for this (just as with initialization) with a number of options:

- `run (NUM STEPS | until CONDITIONS)`

  either runs the simulation a set number NUM of STEPS steps or until a given set of conditions is found to be true (where a condition, we'll soon see, equates to a set of queries on the simulation). STEPS can either be the keyword *step* (which is assumed to be smallest atomic step size) or a given unit of length (assuming the simulation keeps track of time at different levels) (e.g., days, weeks, months, years, centuries, millenia)
- `CONDITIONS := CONDITION (or CONDITION)+`

  CONDITIONS is in disjunctive normal form. In other words it is a number of clauses (CONDITION) each of which consists of queries that are logically connected via *and*, all of which are connected via *or*
- `CONDITION := QUERY (, QUERY)+`

  A CONDITION is a group of queries connected via logical *and* (denoted as a comma ",")
- `QUERY := COUNT entities CONDITION | NUM STEPS`

  A QUERY is either the given number COUNT of entities are found to either have or be lacking the given CONDITION or the NUM of STEPS (as discussed above). Note: COUNT can be:

  ```
  COUNT := NUM | NUM+ | (<|>|<=|>=|==)
   ↪ NUM
  NUM := [0-9]+
  ```

  In other words, the query can test for exact equality or for inequalities of the query counts (greater than, less than, etc.)
- `CONDITION := entities in PATTERN/ARITY| entities (is|isnt|are|arent)? STATUS ((<|>|<=|>=|==) NUM)?| entities (is|isnt|are|arent)? RELATIONSHIP/2 ((<|>|<=|>=|==) NUM)?`

  The possible CONDITIONs are the number of entities that are found in a given PATTERN, the number of entities found with (or without) a given STATUS, the number of entities found with (or without) a given RELATIONSHIP, or the number of STEPS (as discussed above). The CONDITIONs

for STATUS and RELATIONSHIP can optionally do comparisons on the scalar value of the STATUS or RELATIONSHIP. (e.g., "happiness > 5" or "love/2 > 10").

Put together, these options allow for a wide range of different options. For something like *Dwarf Fortress* or *Bad News* where history is generated for a set duration (although *Dwarf Fortress* can be interrupted by the player), it would simply be handled as running for a set number of steps. But for something like the earlier wedding example, it would be inadvisable to run the pre-wedding simulation for a set duration, as there might either be no (or possibly way too many) couples getting married at the end of the duration. Instead, we would want to run until a couple was set to get married:

```
run until 2 entities in pattern
↪ getting_married/2
```

Now, let's consider a tonally different example, where someone has passed away and their loved one(s) have discovered their body:

```
run until 1 entities is recently_deceased,
 1+ entities in pattern discovered_body/2
```

At this stage, we might want to move to a funeral specific ruleset; However, we can imagine that we would want our simulation to be able to handle a range of different large life events separately (e.g., it could look for both weddings and funerals). Putting this together (along with a fixed maximum duration for good measure), we would have:

```
run until 2 entities in pattern
↪ getting_married/2 or
  1 entities is recently_deceased,
  1+ entities in pattern discovered_body/2
  or  20 years
```

As we can see, *Moirai* enables the simulation to run until a set of conditions is true, but we are missing the final piece: handling different scenarios in different manners given those conditions. Putting it all together, we have the final set of operations for *Moirai*, the control flow operations.

3.4. Control Flow

The control flow operations of *Moirai* follow the standard suite of operations (e.g., if, else if, else, while, etc.). The only major departure is that *Moirai* lacks many operations commonly found in most languages (e.g., arbitrary boolean comparisons, arithmetic, assignment), which lends a specific flavor to its handling of the control flow operations. We note that the astute reader might have detected that due to *Moirai's* ability to store/retrieve information and run simulations that the lacking operations are actually latent within it, but we would hope that the astute reader would not implement them just because *Moirai* is Turing complete. The control flow operations are:

- `if CONDITIONS {STATEMENT+}`
  `( (elif|elseif|else if) CONDITIONS`
  `↪ {STATEMENT+} )*`
  `(else {STATEMENT+})?`

  An if block as in many C-style languages; an initial if statement (and subsequent block of code that runs, delimited by curly braces), optionally followed by some number of else if (optionally elif or elsif) statements, with an optional final else. The CONDITIONS are a disjunction of conjunctions as defined as above. A STATEMENT is one of the operations (either previously discussed or a control flow operation) in *Moirai*. The * symbol represents that zero or more of this set of symbols can be seen. The ? symbol means this symbol my appear either zero or one time.
- `while (CONDITIONS|ITERATION (<|<=) NUM)`
  `↪ {STATEMENT+}`
  `ITERATION := (iterations|i)`

  A while loop that runs until some CONDITIONS are met or for a set number of iterations (with many possible terms accepted)
- `yield`

  A yield statement that returns control to the driving program (perhaps to accept user input), but picking up exactly where it left off when *Moirai* is told to run again.
- `choose [ (NUM : {STATEMENT+})+ ]`

  A choice block that randomly chooses one of the inner code blocks to run. NUM is the weight associated with the given block of statements, used to bias the selection away from a uniform distribution.

These control flow operations allow *Moirai* to chain together arbitrary sequences of the operations. For instance, let's return to the previous wedding / funeral example. The different events would have their own rulesets and initialization steps, so we need to be able run the correct simulations:

```
run until 2 entities getting_married/2 or
  1 entities is recently_deceased, 1+
  ↪ entities in pattern
  ↪ discovered_loved_ones_body/2
  or 20 years;
```

```
if 2 entities getting_married/2 {
  ...
}
else if 1 entities is recently_deceased,
  1+ entities in pattern
   ↪ discovered_loved_ones_body/2 {
  ...
}
else {
  ...
}
```

Now, let's say that we wanted this whole thing to run until 5 such events had occurred, it would be as simple as wrapping all of the above in the following block:

```
while iterations < 5 {
    ...
}
```

Moirai enables many different possible modalities of simulations, and in the following section we will discuss different case studies, first progressing though how existing experiences could be implemented with *Moirai* before moving on to other possibilities inspired by non-social simulation based literature, film, and video games.

4. *Moirai* – Case Studies

In the following sections we will first discuss how social simulation has been utilized historically in a variety of experiences. For these cases, we will provide a demonstration of how it could be implemented via *Moirai*. We will then discuss more complex use cases of *Moirai* and the kinds of experience that it can enable.

4.1. Previous Social Simulation Experiences

To begin, we will first examine a number of different social simulation experiences, and their use of social simulation. To provide some vocabulary, we will examine social simulation in various roles:

- **Substrate** – Social simulation that is used **before** the play experience to provide the "world" that the player experiences. Examples include: *Dwarf Fortress*, *Thousand Threads* [41], and *Bad News*
- **Experience** – Social simulation is used **during** play to provide a major (if not sole) component of the play. Examples include: *The Sims*, *Dwarf Fortress*, *Prom Week*, and *Princess Maker*
- **Performance** – Social simulation is used **after** play (if any exists) as a means of providing content to the player (or audience member) to receive. Examples include: *Sheldon County*, *Cozy Mystery Construction Kit* [42] , and the epilogue of *Bad News*

These three categories make up the main delineations in how social simulation has been used up to this point. We note that experiences can use social simulation in multiple categories – e.g., *Dwarf Fortress* uses it as both substrate (history generation) and experience (fortress mode); However, it is common for an experience to use it in only one mode – nearly every "social simulation" game uses it solely as experience (e.g., *The Sims*, *Prom Week*, *Tokimeki Memorial*, *Princes Maker*, etc.). For a visual representation of this continuum see figure 1. We will now examine how these experiences would be set up in *Moirai*.

4.1.1. Social Simulation as Substrate

The simplest example is an experience that uses it solely as substrate. In this way, the world is initialized, it runs, and then stops – at which point the simulation is used as content for some (elided here) downstream process:

```
load simulation.lachesis;
initialize world.clotho;
run 200 steps;
```

4.1.2. Social Simulation as Experience

Not quite as simple is something that uses social simulation solely as a precursor to the core experience, but in large part it looks much the same:

```
load simulation.lachesis;
initialize world.clotho;
while is simulation_still_running {
    run 1 step;
    yield;
}
```

In this case, the simulation runs only one step at a time before yielding to the driving experience to make the game play out as it should.

4.1.3. Social Simulation as Performance

Finally, we come to simulation as performance. However, we will note that it looks (and is) identical to simulation as substrate.

```
load simulation.lachesis;
initialize world.clotho;
run 200 steps;
```

The only difference comes from the framing experience – does the player interact with the simulated world or are they simply a passive audience member?

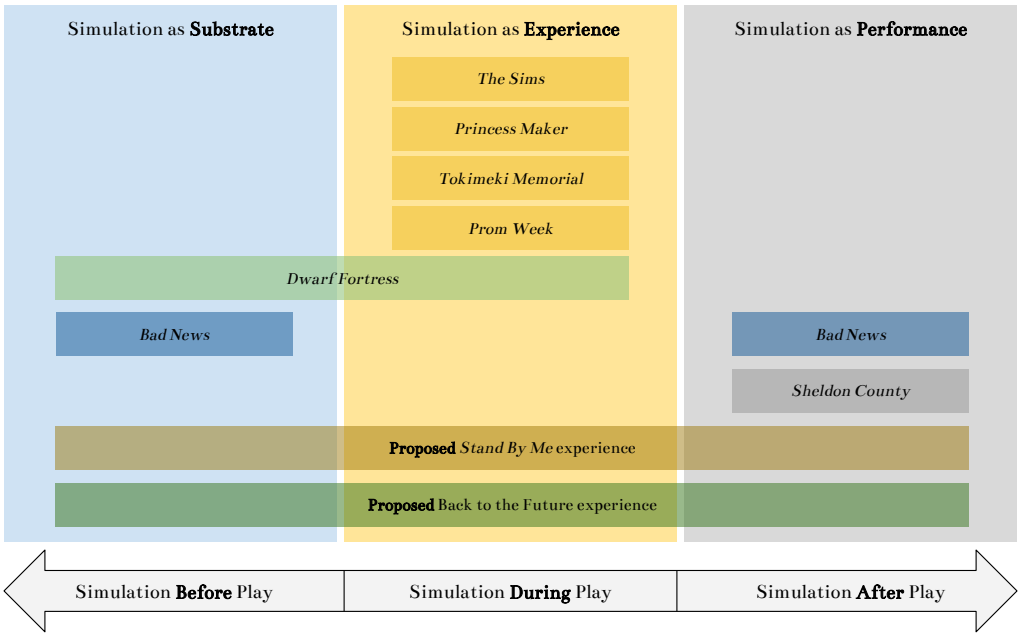

Figure 1: A visualization of different experiences and where they fall on the spectrum of when simulation occurs: **Before**, **During**, or **After** play. Most experiences use the simulation during play (*The Sims*, *Princess Maker*, etc.). Some use it as a substrate to build up history prior to play, but usually use it in additional ways during (*Dwarf Fortress*) or after (*Bad News*) play. To our knowledge, only one experience uses simulation fully for performance (*Sheldon County* [37]). We also propose additional modalities, such as simulation during and after play (to enable epilogues a la *Stand by Me* [39]), or using it to generate history, allowing the player to act within the simulation, and then seeing the ramifications of their actions (to enable time ripples a la *Back to the Future* [40]).

4.1.4. Previous Complex Experiences

As noted in figure 1 there are two noted experiences that use social simulation in various modes – *Dwarf Fortress* and *Bad News* – and we will know show how *Moirai* could be used to drive these experiences.

Dwarf Fortress uses social simulation at a broad level for history generation before transitioning into a more fine-grained moment-to-moment simulation following a dwarven expedition to create the titular mountain fortress. In *Moirai* this would look like:

```
//Substrate
load history_generation.lachesis;
initialize world.clotho;
run 250 years;
//Experience
load dwarf_simulation.lachesis;
initialize fortress.clotho;
while is simulation_still_running {
    run 1 step;
    yield;
}
```

This is just the concatenation of simulation as substrate and experience. *Bad News* used social simulation as both the substrate that the player explored and as performance for an epilogue where the player got to see how life panned out for the characters they encountered in their play through. In *Moirai*:

```
//Substrate
load small_town.lachesis;
initialize small_town.clotho;
run 140 years;
//Experience: Player explores simulated world
yield;
//Performance
run 40 years;
```

Bad News used the same simulation code and the exact same world constructed via simulation for its epilogue, so there is no need for anything other than running during the "performance" portion. In the next section, we will explore a variety of more complex permutations of simulations that are enabled by *Moirai*'s features.

4.2. *Moirai* Enabled Design Patterns

In the following sections we will discuss a number of different design patterns that *Moirai* enables. We will detail the types of experience that the pattern enables and show sample code.

4.2.1. Coarse Simulation to Fine Simulation – Murder Mystery/RomCom

We will start with the simplest mode of combining simulations into an experience, which is similar to the pattern found in *Dwarf Fortress* in which various levels of simulation are chained together. The key difference here is that characters will carry over between the simulations; in *Dwarf Fortress* simulation begins coarsely, with creation-myth style exploits of gods and other fantastical characters who—though not explicitly present in the finer-grained fortress-mode simulation—provide raw material for paintings, songs, and other simulated cultural artifacts.

A large number of tropes in various forms of media often assume some sort of shared backstory and history for the characters that occurred before the portrayed narrative, but carries over and informs the narrative. For instance, a closed room murder mystery such as *Murder on the Orient Express* or *Gosford Park* relies on the characters having intertwined histories such that a large number of characters will have motive for the murder, but the narrative ostensibly occurs after that history has occurred. Similarly, a romantic comedy such as *My Best Friend's Wedding* [43] or *The Proposal* [44] assume some sort of shared history, but the narrative takes place during an important event after this history has transpired (in a way, a closed room romance as opposed to a closed room murder). Unlike *Dwarf Fortress*, characters need to be carried over between the simulations, as opposed to just using the earlier simulation as a broad culture level history generation.

A murder mystery like *Gosford Park* might look like the following, in which an initial coarse simulation runs until several characters have a motive to murder (possibly generating many extraneous characters in its process), and then a secondary finer grained simulation runs involving *only* those characters with motive and others that are close to them:

```
//Coarse history generation
load georgian.lachesis;
initialize georgian.clotho;
run until 5+ characters in pattern
 ↪ have_motive/2;
//Once characters have motive, move to
 ↪ fine-grained closed room
load murder.lachesis;
initialize manor.clotho :keeping entities
 ↪ where in have_motive/2
  :keeping entities where in relationship/2
   ↪ with entity in kept;
run until 1 character is murdered;
```

Similarly, a romantic comedy like *My Best Friend's Wedding* might look like:

```
//Coarse history generation
load modern.lachesis;
initialize modern.clotho;
run until 3+ characters in pattern
 ↪ marriage_love_triangle/3;
//Once a love triangle pops up and 2 are
 ↪ getting married to each other, move to
 ↪ the wedding
load wedding.lachesis;
initialize wedding.clotho :keeping entities
 ↪ where in love_triangle/3
  :keeping entities where in family_of/2 with
   ↪ entity in getting_married/2
  :keeping entities where in good_friend/2
   ↪ with entity in getting_married/2;
run until 1+ characters in
 ↪ romantic_misunderstanding/2;
```

In both cases there is a coarse-grained simulation that runs until some conditions are met, then characters are filtered based on their relationships to each other, and then the final fine-grained simulation occurs until some culminating event has occurred.

4.2.2. Simulate, Modify, Repeat – Back to the Future/Live Die Repeat

A common trope in modern media is that of the timeloop, where a character (or set of characters) is able to relive the same experience over and over such as in the movies *Groundhog Day* [45], *Live Die Repeat* [46] and *Palm Springs* [47]. A related trope is where characters travel into the past where their actions cause changes when they return to the future such as in the movies *Back to the Future* and *The Butterfly Effect* [48]. In both cases, the narrative is reset to a certain point and then changes play out again and again. A *Back to the Future* style experience in *Moirai* would look like:

```
//Coarse history generation
load forties.lachesis;
initialize forties.clotho;
run 10 years;
load fifties.lachesis;
initialize fifties.clotho
        :keeping all characters
        :keeping all locations
        :saving kept as fifties_world;
```

```
while is simulation_still_running {
  while iterations < 7 { //1 week
    yield; //Player interacts
    run 1 day;
  }
  run 30 years;
  yield; //Player sees the changes
  initialize fifties.clotho
    :restoring fifties_world;
}
```

This assumes that there is still substrate that is generated as history, but it is certainly conceivable for that portion to be removed and replaced with a purely authored initial world state. Although the proposed experience above is not identical to the plot of *Back to the Future*, it strikes a similar tone. Here, ten years of "forties world" is simulated as substrate and then recorded as "fifties world" for safe keeping. The player is given a week to experience and influence the entities of fifties world, and then the simulation is run for another thirty years (bringing us to the eighties), to see how these entities ended up. The experience then returns to the stored "fifties world", allowing the player to make different choices, and see what impact those different choices have on the lives of the characters thirty years later as compared to their previous week-long romp in the fifties. As authored above this process can repeat indefinitely, or until some in-game goal criteria is achieved (e.g., ensuring two high-school sweethearts remain in love as they grow older).

4.2.3. Episodic Structure – The Odyssey

Another pattern common in various forms of media is where a core cast of characters encounters different scenarios in an episodic structure. This dates back to at least *the Odyssey* [49], but is also the basis of most modern television programs. Many interactive experiences also build off of this. For instance, the *Persona* [50] series involves a slowly-growing cast of characters that take on problems in a month-based episodic structure. An example in *Moirai* might look like:

```
//Coarse history generation
load history.lachesis;
initialize setting.clotho;
run 2 years;
load episode1.lachesis;
initialize episode1.clotho
  :keeping entities where is main_character;
run until 1+ characters in episode_completed;
load episode2.lachesis;
initialize episode2.clotho
  :keeping entities where is main_character;
run until 1+ characters in episode_completed;
load episode3.lachesis;
initialize episode3.clotho
  :keeping entities where is main_character;
run until 1+ characters in episode_completed;
```

4.2.4. Full Continuum Usage – Stand By Me

One final usage is using social simulation across the full spectrum of before, during, and after play. While this is not as complex as the previous examples, perhaps, it is still an example of social simulation that is enabled by *Moirai* used in a way that has not been implemented in other experiences (to the authors' knowledge). The touchstone for this usage is something like the film *Stand By Me*. In *Stand By Me*, the characters have a shared history, the narrative unfolds and relationships are tested and changes occur, and then the epilogue discusses the lives that the characters had after the main events of the story. In *Moirai* we would envision it as:

```
//Coarse history generation
load history.lachesis;
initialize setting.clotho;
run 2 years;
load main_event.lachesis;
while iterations < 7 { // Play for 1 week
        yield; //Player interacts
        run 1 day;
}
load epilogue.lachesis;
run 25 years;
```

In many ways, this is very much like a combination of *Bad News* (history and epilogue generation) and *Dwarf Fortress* (history generation and active simulation).

5. Conclusion and Future Work

This paper introduces *Moirai*, a Domain Specific Language which facilitates the creation of interactive experiences that leverage social simulation systems. In particular, it is intended to be used for experiences in which simulated narrative content is used before, during, and after play. Through a demonstration of case-studies—Moirai encodings of popular narratives from literature, film, television, and non-simulation games—the authors have shown the versatility of the system, and its capacity for enabling new types of simulation-driven playable experiences that mirror popular tropes and narrative genres in other forms of media.

Natural future work for this project includes building out one or more of these case studies into an actual playable experience. Doing so will enable *Moirai* to be evaluated on its merits both as an authoring assistant and on the quality of the produced experience. This in turn may lead to further insights for language features

that could be added to *Moirai*, leading to greater ease of use and more varied narrative output.

References

[1] H. Jenkins, Game design as narrative architecture, Computer 44 (2004) 118–130.

[2] J. O. Ryan, A. Summerville, M. Mateas, N. Wardrip-Fruin, Toward characters who observe, tell, misremember, and lie, in: Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, 2015.

[3] B. Samuel, J. Ryan, A. J. Summerville, M. Mateas, N. Wardrip-Fruin, Bad news: An experiment in computationally assisted performance, in: International Conference on Interactive Digital Storytelling, Springer, 2016, pp. 108–120.

[4] J. McCoy, M. Treanor, B. Samuel, A. A. Reed, M. Mateas, N. Wardrip-Fruin, Social story worlds with comme il faut, IEEE Transactions on Computational intelligence and AI in Games 6 (2014) 97–112.

[5] B. Samuel, A. A. Reed, P. Maddaloni, M. Mateas, N. Wardrip-Fruin, The ensemble engine: Next-generation social physics, in: Proceedings of the Tenth International Conference on the Foundations of Digital Games (FDG 2015), 2015, pp. 22–25.

[6] J. McCoy, M. Treanor, B. Samuel, A. A. Reed, N. Wardrip-Fruin, M. Mateas, Prom week, in: Proceedings of the International Conference on the Foundations of Digital Games, 2012, pp. 235–237.

[7] D. DeKerlegand, B. Samuel, M. Treanor, Pedagogical challenges in social physics authoring, in: International Conference on Interactive Digital Storytelling, Springer, 2021, pp. 34–47.

[8] M. Guimarães, P. Santos, A. Jhala, Prom week meets skyrim., in: AAMAS, 2017, pp. 1790–1792.

[9] L. Morais, J. Dias, P. A. Santos, From caveman to gentleman: a cif-based social interaction model applied to conan exiles, in: Proceedings of the 14th International Conference on the Foundations of Digital Games, 2019, pp. 1–11.

[10] A. Summerville, B. Samuel, Kismet: a small social simulation language, in: Summerville, A., & Samuel, B.(2020, September). Kismet: a Small Social Simulation Language. In 2020 International Conference on Computational Creativity (ICCC).(Casual Creator Workshop). ACC, 2020.

[11] K. Compton, M. Mateas, Casual creators, in: Proceedings of the Sixth International Conference on Computational Creativity, 2015, p. 228.

[12] G. Smith, J. Whitehead, Analyzing the expressive range of a level generator, in: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, 2010, pp. 1–7.

[13] A. Walker, Marielda 01: The city of light pt. 1, 2016. URL: https://friendsatthetable.net/marielda-01-the-city-of-light-pt-1.

[14] A. Alder, The quiet year (game), Buried Without Ceremony, Pittsburgh, PA (2013).

[15] J. Harper, Blades in the Dark, Evil Hat Productions, 2017.

[16] R. Yeates, Serial fiction podcasting and participatory culture: Fan influence and representation in the adventure zone, European Journal of Cultural Studies 23 (2020) 223–243.

[17] G. McElroy, J. McElroy, T. McElroy, C. McElroy, Ethersea: Episode 1, 2021. URL: https://maximumfun.org/episodes/adventure-zone/the-adventure-zone-ethersea-episode-1/.

[18] G. Gygax, D. Cook, The Dungeon Master Guide, No. 2100, 2nd Edition (Advanced Dungeons and Dragons), TSR, Inc, 1989.

[19] A. Walker, The road to partizan 05: Ech0 and dusk to midnight, 2019. URL: https://friendsatthetable.net/the-road-to-partizan-05-ech0-dusk-to-midnight.

[20] K. Hymes, H. Seyalıoğlu, Dialect: A Game about Language and how it Dies, Thorny Games, 2018.

[21] B. Sovereign, Armour Astir: Advent, Itch.io, 2022.

[22] K. Poh, Ech0, Role Over Play Dead, 2019.

[23] R. Rethal, Dusk to Midnight, Itch.io, 2019.

[24] A. Ramsay, Beam Saber, Itch.io, 2019.

[25] A. Roberts, For the queen [card game], Evil Hat Productions (2019).

[26] B. Robbins, Microscope, Lame M Productions (2019).

[27] T. Adams, Z. Adams, Dwarf fortress, Game [Windows, Mac, Linux], Bay 12 (2006).

[28] J. Whedon, B. K. Vaughan, G. Jeanty, Buffy the Vampire Slayer: Season 8, volume 1, Dark Horse Comics, 2012.

[29] K. J. Wetmore Jr, Uncovering Stranger Things: Essays on eighties nostalgia, cynicism and innocence in the series, McFarland, 2018.

[30] E. Taylor, Dating-simulation games: Leisure and gaming of japanese youth culture., Southeast Review of Asian Studies 29 (2007).

[31] A. Champlin, Playing with feelings: Porn, emotion, and disability in katawa shoujo, Well Played 3 (2014) 63–81.

[32] P. W. Galbraith, Bishōjo games:'techno-intimacy'and the virtually human in japan, Game studies 11 (2011) 31–34.

[33] P. Interactive, Dev diary# 9: Lifestyles, Crusader Kings III 14 (2020).

[34] V. Maxis, V. E. Arts, P. K. Gibson, G. Rodiek, R. M. Vaughan, D. E. Holmberg-Weidler, M. Yang, V. M.

Hollmo, S. Miceli, S. Ross, et al., The sims 4, Electronic Arts (2014).
[35] A. Christie, Murder on the orient express, Lulu. com, 2001.
[36] P. W. Graham, From mansfield park to gosford park: The english country house from austen to altman, Persuasions: The Jane Austen Journal 24 (2002) 211–226.
[37] J. Ryan, Curating simulated storyworlds, Ph.D. thesis, UC Santa Cruz, 2018.
[38] E. Streeter, Father of the Bride, Simon and Schuster, 2015.
[39] R. Reiner, W. Wheaton, R. Phoenix, C. Feldman, J. O'Connell, K. Sutherland, J. Nitzsche, S. King, Stand by me, Skífan, 1986.
[40] R. Zemeckis, Back to the future (film), Amblin Entertainment (1985).
[41] Seamount, Thousand Threads, Seamount, 2022.
[42] M. Kreminski, D. Acharya, N. Junius, E. Oliver, K. Compton, M. Dickinson, C. Focht, S. Mason, S. Mazeika, N. Wardrip-Fruin, Cozy mystery construction kit: prototyping toward an ai-assisted collaborative storytelling mystery game, in: Proceedings of the 14th International Conference on the Foundations of Digital Games, 2019, pp. 1–9.
[43] P. J. Hogan, R. Bass, P. Bosco, C. Diaz, R. Everett, R. Griffiths, J. N. Howard, L. Kovacs, D. Mulroney, J. Roberts, et al., My best friend's wedding, Sony Pictures Entertainment/TriStar, 1997.
[44] A. Fletcher, P. Chiarelli, S. Bullock, R. Reynolds, M. Steenburgen, et al., The Proposal, Touchstone Pictures, 1997.
[45] H. Ramis, T. Albert, D. Rubin, B. Murray, A. MacDowell, C. Elliott, S. Tobolowsky, B. Doyle-Murray, G. Fenton, P. J. Herring, et al., Groundhog day, Columbia Tristar Home Video, 1993.
[46] L. Garcia-Siino, Edge of tomorrow by doug liman, Science Fiction Film and Television 9 (2016) 295–299.
[47] L. Coates, The best new shows, movies to come out of quarantine., UWIRE Text (2020) 1–1.
[48] E. Bress, J. M. Gruber, Butterfly Effect. Director's Cut; Written & Directed By Eric Bress & J. Mackye Gruber; Starring: Ashton Kutcher, Amy Smart, Eric Stoltz, William Scott, Elden Henson, Logan Lerman, Icon Home Entertainment, 2004.
[49] H. Homer, The odyssey, Xist Publishing, 2015.
[50] S. M. Tensei, Persona 4: Golden, 2012.