Game System Models: Toward Semantic Foundations for Technical Game Analysis, Generation, and Design

Rogelio E. Cardona-Rivera,[1,2] **José P. Zagal,**[2] **Michael S. Debus**[3]

[1]School of Computing, University of Utah, Salt Lake City, UT, USA
[2]Entertainment Arts and Engineering Program, University of Utah, Salt Lake City, UT, USA
[3]School of Architecture, Design, and Conservation, Royal Danish Academy of Fine Arts, Copenhagen, Denmark
rogelio@eae.utah.edu, jose.zagal@utah.edu, mdeb@kglakademi.dk

Abstract

Game system models introduce abstractions over games in order to support their analysis, generation, and design. While excellent, models to date leave tacit what they abstract over, why they are ontologically adequate, and how they would be realized in the engine underlying the game. In this paper we model these abstraction gaps via the first-order modal μ-calculus. We use it to reify the link between engines to our *game interaction model*, a player-computer interaction framework grounded in the Game Ontology Project. Through formal derivation and justification, we contend our work is a useful code studies perspective that affords better understanding the semantics underlying game system models in general.

Introduction

There is a wealth of formal models in three broad game design domains: *players*, *interfaces*, and *game systems*. These domains mutually inform and constrain each other (Juul 2007), but relationships between their models remains unclear. We aim toward their unified understanding via a formal *code studies* approach (Aarseth 1997).

To date, game system models are most common and varied. These models represent one or more facets of game design (e.g. mechanics), and computationally support the facet(s) via automated analysis or generation.

However, while models help frame their respective phenomena, they by definition introduce an abstraction gap with the underlying phenomena that is represented.

This gap makes it challenging to use these models – e.g. for game production, research benchmarking – partly because games are often realized via a *game engine*: "modules of simulation code that do not directly specify the game's behavior (game logic) or game's environment (level data)" (Lewis and Jacobson 2002). The gap tacitly obscures the computational foundation that explains what the model's underlying engine logic *is*, how that logic impacts what the model *means*, and how the engine *supports* it (Cardona-Rivera 2020). As a result, it is not trivial to identify what must be implemented to enable a chosen model, nor obvious to know *how* it must be implemented.

We propose a computational framework that codifies the abstraction gap from game engines to game system models. Our framework exposes the depth of abstraction that must be clarified to more-precisely state what a game system model abstracts over, what that means, and how it is realized.

Our contribution is a formal games language grounded in computational science, which models game systems as abstractions over game engines via the *μ-calculus* (Kashima and Okamoto 2008)—a first-order modal logic with *fixpoint operator* μ that can describe the properties of state-transition systems. Our language is summarized via the *game interaction model*, which adequately expresses game systems as described by Zagal's (2014) *game ontology project*, recently unified and expanded in systematic detail by Debus (2019).

Related Work

Researchers (Ebner et al. 2013) and practitioners (Dormans 2009) have called for a common, formal, and conceptual language to better understand video games. This led to much related work we only summarize below. Our work is unique in its combined (a) correspondence to design theory and practice, (b) technology-agnostic formal detail, and (c) breadth.

Game System Models

Game description *languages* (Ebner et al. 2013) and *frameworks* (Grünvogel 2005) are *game system models*—formal representations of game systems that support analysis, generation, and design. They differ in level of abstraction: languages describe systems in terms of *elements*; frameworks break elements down into *properties* and *relations*.

Languages – e.g. Inform (Nelson 2001), VGDL (Schaul 2013), HyPED (Osborn, Lambrigger, and Mateas 2017) – are narrower in scope than our proposal. These support *specific* game genres: interactive fiction design, 2D tiled game generation, and action game analysis, respectively.

In contrast, frameworks are more expressive and closer to us. They allow *creating* languages, thereby supporting *many* game genres. Examples include Dormans' Petri net-based Machinations (2009), Smith et al.'s answer set programming-based LUDOCORE (2010), Martens' linear logic-based Ceptre (2015), Thue and Bulitko's Markov decision processes-based experience manager framework (2018), and Robertson and Young's classical planning-based General Mediation Engine (2018).

What game system models – languages and frameworks alike – have in common is also what distinguishes our work.

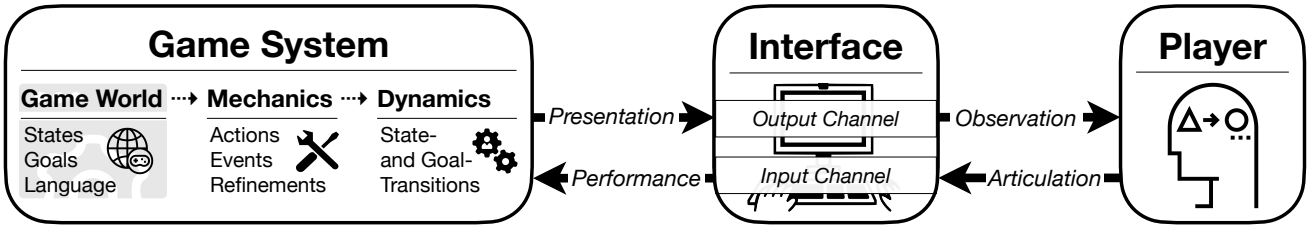


Figure 1: Our proposed game interaction model, a games-variant of Abowd's (1991) human-computer interaction model (whose items are italicized). Solid arrows denote information flow between the Game System, its Interface, and the Player. Boldface items contain more-atomic elements below. Dotted arrows denote information dependencies: Dynamics depend on Mechanics, and in turn on the Game World. We logically derive the Game System from the Game World abstraction over game engines.

Namely: existing models *implicitly* commit to abstractions that afford expressing some games more easily than others.

In languages, it is easier to express games that ontologically rely on elements provided by the language's abstractions. In frameworks, it is likewise easier to support games that ontologically rely on properties and relations provided by the framework's abstractions. In contrast, we develop a framework that directly examines these abstractions.

One exception that makes commitments more explicit is work we take inspiration from: the framework of operational logics (OLs). An OL represents an *ensemble* of elements *across* game systems, interfaces, and players (Osborn, Wardrip-Fruin, and Mateas 2017, p. 2, emphasis added):

> [OLs] *tie together* ***low-level and abstract mechanics*** [in the game system] *with the* ***presentation of sensory stimuli*** [via an interface] *to* ***players*** *to enable an understanding of what the game is doing and how it functions.*

These ensembles are computationally-realizable in many ways, hence their pluralized framing (e.g. camera logics, collision logics). But as noted by Martens (2015, p. 51): "We lack good specification tools for inventing new operational logics, or combining elements from several." Our framework embodies this systematicity and compositionality—akin to OLs in overall aims, yet complementary to them. Instead of modeling game design phenomena that *spans* game systems, interfaces, and players as OLs do, we conceptualize these as distinct environments with properties and relationships that afford building games via their principled orchestration.

Trends in Game Description Frameworks

Our work is distinct from game description frameworks to date in underlying game ontology and abstraction rationale.

Underlying Game Ontology Modern frameworks appeal to intuitions of what is ontologically important to the games they support, but what led to said intuitions is unfortunately left unstated (Aarseth and Möring 2020). As noted by Osborn (2018, p. 8–9):

> *Other approaches to describing game designs have ontological issues as well as pragmatic ones.* [e.g.] *the Game Ontology Project* [...] *decompose*[s] *games with base terms that come from effectively infinite grammars* [...] *This would mean* [one] *would have to complete a huge ontology-development project before being able to do any serious work.*

Ontology development is non-trivial, and we do not critique the pragmatic choices made for developing OLs or other frameworks. But in contrast to prior work, we explicitly build toward ontologies developed over several decades within game studies as unified by Debus (2019).

Abstraction Rationale Frameworks to date *start* with the base assumption that their respective formalism – Petri nets, answer set programming event calculi, linear logic, Markov decision processes, planning, etc. – is *epistemologically adequate* (McCarthy 1981) for reasoning over the genres of games of interest to the modeler. However, no framework has been accompanied by formal argumentation that *justifies* such adequacy for its level of abstraction.

While this does not preclude a framework from being useful it does beg the question that motivates and differentiates our work: what *are* current game system models abstracting over? We explore one answer in this paper, using a *pragmatic* abstraction rationale. That is, our game interaction model abstracts over a novel *game engine* model we develop, itself built from elements that: (a) we minimally need to represent to eventually support reasoning over Debus' unified ontology of games, and (b) feature as common enough across game engines to warrant their representation.

Overview and Background

We develop the *game interaction model*, a framework to model player-computer interaction in terms of game systems, players, and interfaces between them (Fig. 1). We focus on the game system, complementing Martens and Hammer (2017), who model the interface from player to system.

We model the game system in three stages. First, we develop a formal model of game engines. Second, we derive an abstraction of engines via a parameter-space partition that results in a model of the *game world*: the bridge between engines and systems. Third, we develop the rest of the game system model around the game world by representing *mechanics* and *dynamics* that the world supports. Our modeling work relies on the concepts and formalisms we detail next.

Concepts

Our game interaction model is a games variant of Abowd's (1991) formal *human-computer interaction model*.

Abowd frames HCI as a four-stage cycle: (1) a user *articulates* a desired action via an interface, (2) the interface

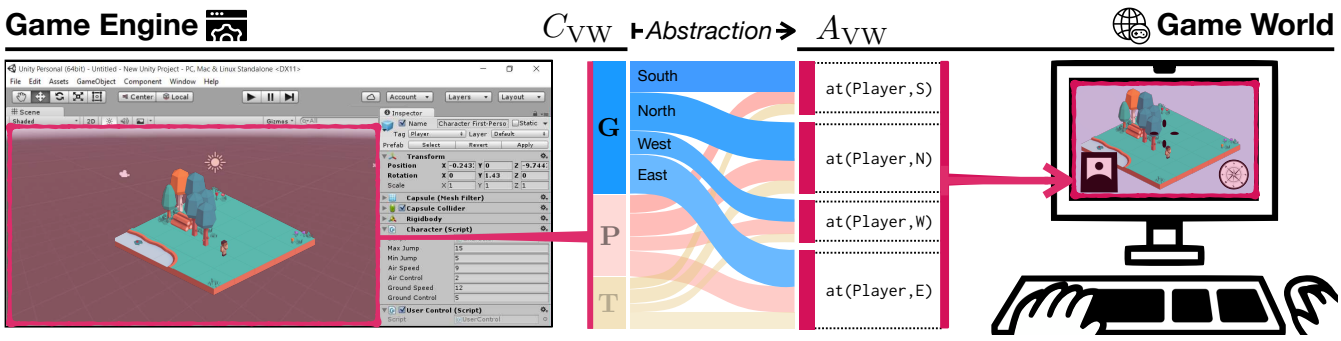


Figure 2: We model a game engine as a special C_{VW} that includes geometry $\mathbf{G}$, physics $\mathbf{P}$, and time $\mathbf{T}$. An A_{VW} is an abstraction that partitions a C_{VW} to identify "what ludologically counts" as the Game World; e.g., what locations a `Player` can be `at`.

Goal Type	Description
Ultimates	("Games that...")
Win	Effect an evaluation when an end state is reached
Finish	Effect no evaluation when an end state is reached
Prolong	Conclude against designer or player intent
Imperatives	("Players must...")
Choose	**Select** an element from a finite set
Configure	**Manipulate** elements into a "correct" state
Create	**Bring** a new **entity** into existence
Find	**Locate** a particular **entity**
Obtain	**Bring** a particular **entity** under control
Optimize	**Accumulate** a requested amount of a particular **entity**
Reach	**Navigate** to a particular **location**
Remove	**Eliminate** an existing **entity**
Solve	**Select** a "correct" element from a finite set
Synchronize	**Bring** at least two elements into **spatiotemporal** unity

Table 1: Ultimate goals determine a game's end. Imperative goals are closer-to-gameplay, and required to satisfy an overarching ultimate. Boldface words are space, time, entity, and mechanics concepts present in the UGO's goal typology.

translates input into code that possibly *performs* a system state change, (3) the system *presents* the resulting change or diagnostic information to the user, and (4) the user updates their mental model by *observing* the interface. To us, the users are players, and the systems are games.

In turn, the game system model is a computational rendition of the Unifying Game Ontology (UGO, Debus 2019), a classification of game elements across six facet categories and three properties. In the UGO, games are artifacts that represent *space*, *time*, *entities*, *mechanics*, *goals*, and *randomness* facets. Our prior work highlights how goals are key (2020): the UGO's typologies for space, time, entities, and mechanics all manifest within the goals one (see Table 1). Finally, each facet may manifest *explicitly* or *implicitly*, *statically* or *dynamically*, and *discretely* or *continuously*.

Formalisms

We derive the game system model as an abstraction over a game engine represented as LaValle's (2019) *virtual world* (VW): a computational model of stimuli to render. We argue there are two such worlds to distinguish (Fig. 2). The *concrete* one is defined by the engine. The *abstract* one is defined by the game designer, and we term it the *game world*.

To go from concrete to abstract, we use Cardona-Rivera's (2020) theory of game design abstraction as *sound approximation*: each VW is a partially-ordered set of parameters, and represents a different layer of information abstraction. The *concrete virtual world* C_{VW} is the more-quantitative one, and it is linked to the more-qualitative *abstract virtual world* A_{VW} via a *scheme*: $\Upsilon = \langle C_{\text{VW}}, f_C, A_{\text{VW}}, f_A, \mathcal{L}, \mathcal{G} \rangle$.

An abstraction scheme is motivated when the reasoning to perform via $f_C \colon C_{\text{VW}} \to C_{\text{VW}}$ (e.g. pathfinding in Euclidian space) is not efficiently computable. A scheme's *lifting function* $\mathcal{L}$ maps information in the C_{VW} onto the A_{VW}, where the more-qualitative reasoning $f_A \colon A_{\text{VW}} \to A_{\text{VW}}$ (e.g. A^*) is an *order-preserving* (i.e. sound) proxy for f_C. When f_A's result is mapped via the *grounding function* $\mathcal{G}$ back into C_{VW}, there is a loss of information because $|C_{\text{VW}}| > |A_{\text{VW}}|$.

Our C_{VW} models a game engine as a special labeled state-transition system (S, U, γ), where S is a set of states, U is a set of labels, and $\gamma \subseteq S \times U \times S$ is a transition relation.

In turn, our A_{VW} models game worlds as an abstraction over the C_{VW} via STRIPS (Fikes and Nilsson 1971), as used for *classical planning problems*: search problems over dynamically constructed state-transition systems. STRIPS depends on L—a set of sentences drawn from a logic language.

The Logic Basis of A_{VW} In STRIPS, L is typically drawn from a first-order language (FOL). However, our A_{VW} reasons over a C_{VW} that encodes a state-transition system; doing so requires *modal* reasoning not afforded by FOL. We thus adopt the FO modal logic μ-calculus (FOM$_\mu$)—a modest extension of FOL that can describe state-transition system properties (Kashima and Okamoto 2008).

FOM$_\mu$ formulas are defined in Backus Naur form as:

$$\phi ::= \bot \mid \top \mid \mathtt{p}^n(\mathtt{t}_1, \ldots, \mathtt{t}_n) \mid \neg\phi \quad (1)$$

$$::= \phi \wedge \phi \mid \phi \vee \phi \mid \phi \to \phi \mid \phi = \phi \mid \forall x.\phi \mid \exists x.\phi \quad (2)$$

$$::= X \mid [u]\phi \mid \langle u \rangle \phi \mid \mu X.\psi \quad (3)$$

Eqs. 1 and 2 are the literals and formulas of FOL, respectively; $\mathtt{p}^n \in P$ is an n-ary predicate symbol and $\mathtt{t}_i$ is a term symbol. Terms can be constant symbols C, n-ary function symbols $f^n \in F$, or variable symbols $x \in V$.

Eq. 3 are FOM$_\mu$-specific formulas, defined over a *structure* $K = \langle S, U, \gamma, \mathcal{D}, \mathcal{I}_\mu \rangle$. S, U, and γ define a ***Kripke*** state-transition model of worlds (S) and accessibility relations (U, γ). Each $s \in S$ defines a *local* discourse domain, and $\mathcal{D}$ is the union of all of them. $\mathcal{I}_\mu$ is an *interpretation*—a mapping from 2^S to $\mathcal{D}$. In (3), X is a *propositional* variable, $u \in U$, and ψ is a formula where every free X is positive.

Unlike (term) variables $x \in V$, propositional variables $X \in \mathcal{V}$ bind to *formulas* ϕ; i.e. X denotes a truth value. Further, $[u]\phi$ denotes that ϕ holds in all states reachable via u from the state in which it is asserted. Its dual $\langle u \rangle \phi$ denotes that ϕ holds in at least *one* such state. Finally, the fixpoint formula $\mu X.\phi$ denotes the X least fixpoint of ϕ, true in the smallest subset of states $S_{\min} \subseteq S$ whereby substituting with $S_{\min}$ all the free X in ϕ—denoted $\phi(X)$—results in what is true of X being true of $\phi(X)$, i.e. $X = \phi(X)$.

For a set of ϕ asserted in a given s, the interpretation $\mathcal{I}_\mu$ gives the semantics of its symbols by mapping:
(1) every $c \in C$ and $x \in V$ onto an element from $\mathcal{D}$ each,
(2) every $f^n \in F$ onto a domain function $\mathcal{F}\colon \mathcal{D}^n \to \mathcal{D}$,
(3) every $\mathsf{p}^n \in P$ onto a domain tuple $\mathcal{P} \subseteq \mathcal{D}^n$, and
(4) every $X \in \mathcal{V}$ onto states $S_X \in 2^S$ in which X is true.

The Planning Basis of A_{VW} We use the FOM$_\mu$ language to define a STRIPS problem $\Sigma = \langle L, S, \gamma, A, E, I, g \rangle$.

$S \subseteq 2^L$ is a family of states; each $s \in S$ is a set of positive FOM$_\mu$ sentences denoting what is true in s; sentences not in s are false. A label in U defines a *transition*—a triple $\langle \text{PRE}(u), \text{ADD}(u), \text{DEL}(u) \rangle$, of precondition, add, and delete lists respectively, all subsets of 2^L. If a transition u is *executed*, the resulting state is given by γ—the function:

$$\gamma(s, u) = \begin{cases} (s \setminus \text{DEL}(u)) \cup \text{ADD}(u) & \text{PRE}(u) \subseteq s \\ s & \text{otherwise} \end{cases} \quad (4)$$

Transitions under the planning agent's control are its actions $A \subseteq U$; events $E \subseteq U$ are all other transitions. Given an initial state $I \in S$, the agent must synthesize a sequence of actions whose execution manifest the goal conditions $g \subseteq 2^L$. These conditions implicitly define a family of states S_g; namely, those that satisfy all conditions: $S_g = \{s \mid (s \in S \in \Sigma) \wedge (g \subseteq s)\}$.

By default, a STRIPS agent cannot systematically adopt, revise, or drop its goals. This has motivated research on *goal reasoning*, which seeks to endow agents with said capacity.

Given the primacy of goals in games, we represent a *goal transition system* as a tuple $\Theta = \langle \Sigma, G, R, \beta \rangle$ per Cardona-Rivera et al. (2022). Here, Σ is a STRIPS problem as before. The set G encompasses all possible goal state families S_g. Particular goal conditions can change in a given state based on refinements in R. A refinement is isomorphic to a transition and yields a goal per the goal-transition function β:

$$\beta(g, s, r) = \begin{cases} (g \setminus \text{DEL}(r)) \cup \text{ADD}(r) & \text{PRE}(r) \subseteq s \cup g \\ g & \text{otherwise} \end{cases} \quad (5)$$

Problem Statement

We offer a solution to the open problem of modeling game systems at a level of abstraction that is formally justified as (a) ontologically adequate to represent games, and (b) epistemologically adequate to reason over them.

Approach We justify ontological adequacy by supporting the UGO, justified in turn via theoretical saturation. We justify epistemological adequacy by formally deriving how our model may be realized in a game engine. Unlike prior work, our derivation is agnostic to an implementing engine because it is based on a mathematical game engine model.

Constructing the Game World Abstraction

In this section, we reify the abstraction gap by representing a game engine as a C_{VW}, using the Unity Game Engine's Entity–Component System (ECS) as a running example. We then formally derive how an engine is linked to the game world component (from Fig. 1).

A Formal Model of Game Engines

Game engines are built as a collection of interfaces that expose all the parameters a game designer has available; e.g. several of Unity's parameters appear in its user interface (Fig. 2). To model game engines mathematically, we rely on the C_{VW}—it represents the smallest set of independently manipulable and computationally realizable parameters a designer may use to build games.

We propose a C_{VW} is implicitly defined by a game engine. An engine imposes a particular *family* of parameters; each family member computationally models some facet of the real world (cf. Virtual Reality, LaValle 2019). As such, there is not a *single* C_{VW}; each engine defines its own. At the same time, our community-of-practice has converged on at least three "fundamental abstractions"—family members that feature in a wide array of engines (Gregory 2018).

The **graphics** ($\mathbf{G}$) engine abstraction is a computational model of geometric information that defines a game's spatial universe as well as the spatial form of entities within it. Unity requires that every *GameObject*—i.e. Entity to be rendered—define a *Transform* Component, "used to store and manipulate the position [and] rotation [...] of the object" (Unity Technologies 2021, cf. Transform). It represents the 3D Euclidian Space $\mathbb{E}^3$ and geometric constructs with *position*—$\vec{p} = (x, y, z) \in \mathbb{E}^3$—and *orientation* within $SO(3)$, the set of all 3D rotations for p—described as unit quaternions $\vec{q} = (1, \mathbf{i}, \mathbf{j}, \mathbf{k}) \in \mathbb{R}^4$.

The **physics** ($\mathbf{P}$) engine abstraction is a computational model of physical information that defines geometric behavior with respect to motion. Unity requires a *Rigidbody* Component for a GameObject to be put under the control of its physics engine, which "lets you apply forces to the object and control [it] in a physically realistic way" (Unity Technologies 2021, cf. Rigidbody). It represents reasoning about *translation* (changes in position) and *rotation* (changes in orientation) per *linear velocity* ($\mathbf{V}$) and *acceleration* ($\mathbf{A}$), and their *angular* counterparts ($\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$), all subsets of $\mathbb{R}$.

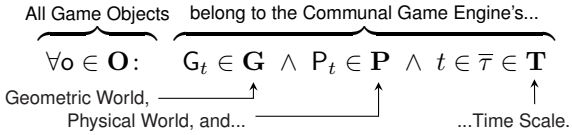
The **time** ($\mathbf{T}$) abstraction is a computational model of temporal information that manifests via the *game loop*, which defines the sequential dynamics of the game. In Unity, clicking on the Play button ▶ allows an Entity's Component values to change (Unity Technologies 2021, cf. Time). By default, the loop runs at 60Hz over floating point ($\approx \mathbb{R}$) real world time; i.e. we observe one game *frame*—game engine snapshot of all GameObjects—over $\Delta t = 1/60$ seconds.

The abstractions $\mathbf{G}$, $\mathbf{P}$, and $\mathbf{T}$ are so common that we say they form part of a *communal game engine*—a special C_{VW}:

Definition 1 (Communal Game Engine). A model of a game engine, defined as a tuple $C'_{\text{VW}} = (\mathbf{G}, \mathbf{P}, \mathbf{T}, \mathbf{O})$. $\mathbf{G} = \big(\mathbb{E}^3, SO(3)\big)$ is a *geometric world*. $\mathbf{P} = (\mathbf{V}, \mathbf{A}, \boldsymbol{\omega}, \boldsymbol{\alpha})$ is a *physical world* of unit-mass rigid-bodies. $\mathbf{T} = (\overline{\tau}, \sigma)$ is a time scale, where $\overline{\tau} \subseteq \mathbb{R}$ is the *time domain* and σ is the *forward jump operator* over a *time point* $t \in \overline{\tau}$.

$\sigma(t)$ finds the next time point: $\sup\{t' \mid t' \in \overline{\tau} \wedge t' > t\}$. We define it as $\sigma(t) = t + \Delta t$ with $\Delta t = 1/60$.

Finally, $\mathbf{O}$ is a set of *game objects*. Each game object is a tuple $\mathsf{o} = (\mathsf{G}_t, \mathsf{P}_t, t)$, where G_t is its *geometric state* at t, P_t is its *physical state* at t, and t is its *time index*, s.t.:



An o's *state* at t is $\boldsymbol{\Phi}(\mathsf{o}, t) = (\vec{p}_t, \vec{q}_t, \vec{v}_t, \vec{a}_t, \vec{\omega}_t, \vec{\alpha}_t, t)$. In turn, a C'_{VW}'s *state* at t is the state of all its objects:

$$\boldsymbol{\Phi}(C'_{\text{VW}}, t) = \bigcup_{\mathsf{o} \in \mathbf{O}}^{C'_{\text{VW}}} \{\boldsymbol{\Phi}(\mathsf{o}, t)\} \tag{6}$$

Eq. 6 demonstrates how the C'_{VW} is a *discrete time dynamic simulation* (DTDS, García and Mollál 2005), whose temporal evolution can be modeled via a state-transition system. We distinguish the C'_{VW} system as *concrete*: $(\mathbf{S}, \mathbf{U}, \boldsymbol{\gamma})$.

It defines $\mathbf{S}$ from all possible values in Eq. 6, $\mathbf{U}$ from all possible changes to values of a t-indexed state $\boldsymbol{\Phi}(C'_{\text{VW}}, t)$, and $\boldsymbol{\gamma}$ as a state mapping from t_i to t_{i+1} through a transition. Concrete states and transitions ($\mathbf{S} \times \mathbf{U}$) define the C'_{VW}'s *phase space*, and we define $\boldsymbol{\gamma}$ as the commonly used Newton-Euler-1 method (see Gregory 2018; LaValle 2019).

From Game Engines to Worlds via Goals

We define goals as conditions in a C_{VW} that players are expected to meet during play. As we prove in Theorem 1, goals *necessarily* introduce a level of game engine abstraction. The intuition is that designers must minimally specify what "counts" to meet a game's goals. This indirectly specifies what does *not* count, meaning that we *at least* distinguish between (more-abstract) goal v. non-goal states.

For example, imagine we are designing a game where the goal is to reach one of four cardinal locations (Fig. 3).

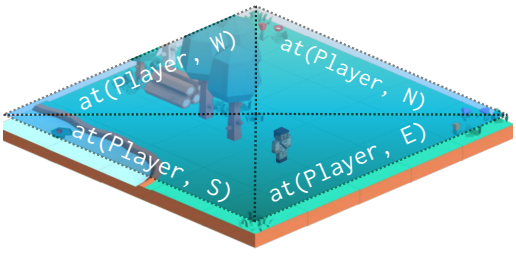


Figure 3: The goal state of being `at` one of four locations induces a partition of C'_{VW}'s geometric world, as illustrated.

To do so we must define: (1) what phase *subspace* counts as part of each cardinal location, and (2) what it means for the player to reach one. This partitions our C'_{VW} into four geometric subspaces, and distinguishes two states: the state of being `at` the goal location, and the state of *not* being there.

Theorem 1 (Goal State Abstraction). Defining a concrete goal state for a C_{VW} induces an abstraction over it.

Lemma 1.1 (Goal State Partitioning). Defining a concrete goal state for a C_{VW} induces a partition over the C_{VW} set.

Proof sketch, Lemma 1.1 (Contradiction of Cases). Assume we define a concrete goal state $\mathbf{s}_g$ that does *not* induce a partition over the set C_{VW}. Then, either: (a) all elements of C_{VW} are part of $\mathbf{s}_g$, or (b) no elements of C_{VW} are part of $\mathbf{s}_g$; otherwise, there would be a partition. In (a): if all elements are part of $\mathbf{s}_g$, then all conditions are trivially met and there is no game, and thus there is no $\mathbf{s}_g$ ($\perp$). In (b): if no elements are part of $\mathbf{s}_g$, then $\mathbf{s}_g$ does not exist ($\perp$). □

Proof sketch, Theorem 1 (Direct). Per Ranzato and Tapparo (2007), defining a set partition induces an abstraction over the set. Because a concrete goal state induces a set partition over C_{VW} (Lemma 1.1), it induces an abstraction over it. □

The result of goal state abstraction is our *game world*, where subsets of the C_{VW} reify as elements of the A_{VW}.

The A_{VW}'s Time Scale Abstracting over C'_{VW} produces a side-effect: because C'_{VW} is a DTDS (Eq. 6), *any* resulting A_{VW} is *also* a DTDS, but at a *possibly coarser time scale* (Giambiasi and Carmona 2006). This was described by Zagal and Mateas (2010) as going from *game world time*—induced by events in the world, measured in an engine's *discrete wall-clock*—to *coordination time*—induced by events that coordinate agents, measured in *turns* or *actions*.

We describe this more precisely: whereas the C'_{VW} operates in game world time as recorded in the engine's time scale $\mathbf{T}$, the A_{VW} operates in coordination time as recorded in the *game world's time scale*—we denote it as $\mathbb{T} = (\dot{\tau}, \delta)$. Thus, the C'_{VW} and A_{VW} define their own transition systems, which are linked via their respective time scales. In the limit, the A_{VW}'s scale can match but not exceed the density of the C'_{VW}'s: i.e. $\dot{\tau} \subseteq \overline{\tau}$ (García and Mollál 2005).

For example, coordination time might operate over seconds to enforce *timed turns* in our game. This would relate $\mathbb{T}$ to $\mathbf{T}$ as in Fig. 4, effectively making $\dot{\tau}$ the integers $\mathbb{I}$.

To stay linked from onset—when $(t_0 \in \overline{\tau}) = (\mathsf{t}_0 \in \dot{\tau})$—the δ operator must preserve the order of σ (García and Mollál 2005). Thus, $\mathbb{T}$ *must* soundly approximate $\mathbf{T}$.

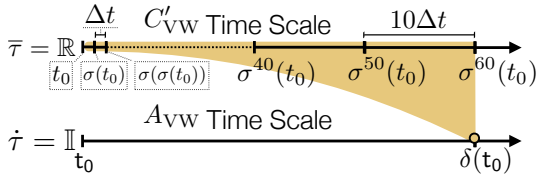


Figure 4: When the C'_{VW}'s time scale $\overline{\tau} = \mathbb{R}$ is abstracted into the A_{VW}'s time scale $\dot{\tau} = \mathbb{I}$, 60 applications of $\sigma(t)$ "fit" in 1 application of (the coarser) $\delta(t) = \lceil t \rceil$ ($\because 60\Delta t = 1$).

A Formal Model of Game Worlds

Much like there is not *one* C_{VW}, there is also no *one* A_{VW} that may be defined atop a given one.

For instance, in our previous example from Fig. 3 we *chose* to take the C'_{VW}'s geometric world and partition it to suit our design needs: i.e., our need to denote the cardinal locations in order to state when a player is `at` one. But in general, a game designer may construct any number of A_{VW}s for a given C_{VW}. How to go about building one depends on what goals a designer wishes to represent. We could have wanted to represent cardinal *and* ordinal locations (i.e., `Northwest`, `Southwest`, `Northeast`, `Southeast`), or a different goal altogether. This would impact what the A_{VW}'s reified subsets mean beyond what the contained elements from the C_{VW} intrinsically represent (which for the C'_{VW} are numbers). Had we represented the ordinals for example, a portion of `North`'s subspace would cease to mean North and would instead mean Northwest (or Northeast).

Further, in the same way that a C_{VW} is implicitly defined by the engine, an A_{VW} is implicitly defined by whatever is *built* with the engine. That is, a technical game designer need not define the A_{VW} overtly; it will manifest as a consequence of game programming. The mentioned concept of being `at` a location, for example, is indirectly introduced by the code in Listing 1. This code performs part of the abstraction illustrated in Fig. 3, allowing a designer to check whether a player is contained within the `Mesh` (i.e. geometric world subspace) that bounds a cardinal location.

```
1 Transform player; // Player's geometric state
2 Dictionary<string, Mesh> cardinals; // N,W,S,E
3 bool At(string name) {
4  Mesh loc = cardinals[name]; // assume ok
5  if(! loc.bounds.Contains(player.position))
6     return false; // Player not at given cardinal
7  return true; } // Player at given cardinal
```

Listing 1: Code to check a `Player` being `at` a `location` (Unity `Mesh`), indirectly introducing part of the abstraction in Fig. 3.

Expanding the Communal Model Deriving an A_{VW} from our communal game engine led to our introduction of $\mathbb{T}$. One may naturally ask whether abstract virtual worlds exhibit enough regular structure to warrant introducing $\mathbb{G}$, $\mathbb{P}$, and $\mathbb{O}$ as analogues to $\mathbf{G}$, $\mathbf{P}$, and $\mathbf{O}$. *Is* there a *communal game world* A'_{VW} worth defining? We believe there is.

The *only* requirement that such an A_{VW} must satisfy is that it be a sound approximation of its C_{VW} (Cardona-Rivera 2020). This means that modeling the communal game world atop the C'_{VW} only requires the A'_{VW} to: (1) represent a state-transition system that soundly approximates the one defined by C'_{VW}. While $\mathbb{T}$ with analogous $\mathbb{G}$, $\mathbb{P}$, and $\mathbb{O}$ offers an epistemologically adequate representation for an A'_{VW}, other formalisms may do so as well. We therefore turn to our guiding pragmatic rationale: the desire for ontological adequacy.

For us, justifying our communal game model as ontologically adequate is tantamount to ensuring that the A'_{VW} is capable of supporting the UGO as discussed before. This adds two additional modeling constraints; namely, that the A'_{VW} can: (2) represent the goal typology from Table 1, and (3) link to other UGO facets as the goal typology demands.

In service of the above three modeling constraints, we propose a general-purpose and parsimonious approach to representing the A'_{VW}—via a FOM$_\mu$-backed STRIPS-based representation. Using FOM$_\mu$ affords reasoning about the C'_{VW}'s concrete state-transition system at the A'_{VW} level, and using STRIPS guarantees that the A'_{VW} itself codifies its own state-transition system. In effect, Def. 2 "lifts" $(\mathbf{S}, \mathbf{U}, \boldsymbol{\gamma})$ to a coarser-grain transition system. Formally:

Definition 2 (Communal Game World). An abstraction of the C'_{VW}, defined as a tuple $A'_{\text{VW}} = \langle L, S, U, I, G \rangle$.

L is a set of FOM$_\mu$ sentences over a *concrete* structure $K = \langle \mathbf{S}, \mathbf{U}, \boldsymbol{\gamma}, \mathcal{D}, \mathcal{I}_\mu \rangle$, where $\mathcal{I}_\mu \colon 2^{\mathbf{S}} \to \mathcal{D}$ and $\mathcal{D} = C'_{\text{VW}}$.

$\langle S \subseteq 2^L, U \rangle$ are *abstract* states and transitions, $I \in S$ is the game world's initial state, and $G \subseteq 2^S$ is the set of possible goal state families for the game world.

Technical game design requires "lifting" the C'_{VW} into an A'_{VW}, and precisely defining how operations at the A'_{VW} "ground out" in the C'_{VW} it abstracts. Def. 2 supports a *communal* abstraction scheme $\Upsilon'_{\supset}$ in that regard:

$$\Upsilon'_{\supset} = \Big\langle \overbrace{C'_{\text{VW}}}^{\text{Def. 1}}\ ,\ \overbrace{\boldsymbol{\Phi}}^{\text{Eq. 6}}\ \Big|\ \overbrace{A'_{\text{VW}}}^{\text{Def. 2}}\ ,\ \overbrace{\gamma \times \beta}^{\text{Eqs. 4 and 5}}\ ,\ \underbrace{\mathcal{L}}_{\text{The introduction of goals (see Theorem 1).}}\ ,\ \underbrace{\mathcal{I}_\mu}_{\mathcal{I}_\mu \colon 2^{\mathbf{S}} \to \mathcal{D}} \Big\rangle \quad (7)$$

(Defined by the Game Engine; Defined by the Technical Game Designer)

Because A'_{VW} must soundly approximate C'_{VW}, $\boldsymbol{\Phi}$ and $(\gamma \times \beta)$ must be *bisimilar* ("match each other's moves"). Bisimulation can be proven via FOM$_\mu$, but is out of scope.

A Formal Model of the Game System

Def. 2 contextualizes the rest of our game system model, which is summarized in Table 2 and detailed as follows.

Definition 3 (Mechanics). A model of world transitions actionable by player and non-player agents, invariant under perception, and definable relative to *states* and *goals* (see Lo et al. 2021), defined as a tuple $M = \langle A, E, R \rangle$.

$A \subseteq U$ are *player actions*. $E \subseteq U$ are *non-player events*. R are *player goal refinements*, actionable by all game agents.

Definition 4 (Dynamics). A model of run-time behavior of world mechanics acting on player inputs, system responses, and trajectories thereof over time (cf. Hunicke, LeBlanc, and Zubek 2004), defined as a tuple $D = \langle \gamma, \beta \rangle$, where these are transition functions akin to Eqs. 4 and 5, defined in Eq. 10.

Concept	Constructs
Game World	States (VR, AP), Goals (GR), Language (μ)
Mechanics	Actions (AP), Events (AP), Refinements (GR)
Dynamics	State- (AP) / Goal-transitions (GR)

Table 2: We map game system model concepts (left) to formal constructs (right) from automated planning (AP), goal reasoning (GR), virtual reality (VR), and the μ-calculus.

Definition 5 (Game System). A model of game artifacts, defined as a tuple $\eth = \langle A'_{\text{VW}}, M, D \rangle$.

A'_{VW} is the *game world* (Def. 2), M are the *mechanics* (Def. 3), and D are the *dynamics* (Def. 4). $\eth$ is defined such that game transitions are the mechanics,

$$U \equiv A \cup E \cup R \tag{8}$$

these transitions are defined per the A'_{VW}'s FOM_μ language,

$$\forall u \in U \colon \{\text{PRE}(u), \text{ADD}(u), \text{DEL}(u)\} \subseteq 2^L \tag{9}$$

and dynamics are defined over the world, its mechanics, and the history of state- and goal-transitions that may manifest.[1]

$$\gamma \colon 2^S \times 2^U \to S \text{ and } \beta \colon G \times 2^S \times 2^R \to G \tag{10}$$

Summative Assessment and Future Work

We review in turn how our work supports our claims that (a) we expose the depth of abstraction latent in current game system models, (b) our game system model is ontologically adequate relative to the UGO, and (c) our game system model is epistemically adequate relative to our code-agnostic justification of realizability via a game engine.

Uncovering Latent Abstractions in Game System Models Our derivation demonstrates that the depth of abstraction in game system models to date is tantamount to the depth of abstraction between *operational* and *denotational semantics*.

Operational concerns are proof-theoretic and give meaning to models in terms of algorithms that identify which outputs are obtainable from which inputs. Denotational concerns are truth-theoretic and give meaning to models in terms of algorithm-agnostic mathematical functions.

Operational semantics have dominated game system models to date, which support specifying languages and interpreters to abstractly reason over games. Complementarily, denotational semantics captures the mathematical meaning that underpins a program written in such a language, and in this paper we define a mathematical language for game system models. We agree with Scott (1970, p. 169) that:

> *It is all very well to aim for a more "abstract" and a "cleaner" approach to semantics, but* [...] *in the end the* [model] *still must be run on a machine* [...] *unless there is a prior, generally accepted mathematical definition of a language at hand, who is to say whether a proposed implementation is correct?*

Ontological Adequacy for Games We support the UGO's properties as follows. A facet is *dynamic* if it is in the domain of γ and β, and is *static* otherwise. Further, a facet is more-*continuous* the more it manifests in the C'_{VW}, and more-*discrete* the more it manifests in the A'_{VW}. Critically, our formalisms only represent *explicit* facets because by definition the *implicit* ones appeal to "a person's cognitive capabilities" (Debus 2019, p. 150). Thus, being implicit is *not* a game system property; it is player-relative.

Of the UGO's facets, two are overtly reified: *mechanics* via Def. 3, and *goals* via Def. 2. The *space*, *time*, and *entity* facets manifest in both C_{VW} and A_{VW}; as $\mathbf{G}$ and $\mathbf{P}$, $\mathbf{T}$, and $\mathbf{O}$ in the former, and via L-encoded abstractions in the latter. The *randomness* facet requires a small revision to Eq. 10. Our discussion simplified it as deterministic. To accommodate stochasticity, we set the range of γ as 2^S and that of β as 2^G; i.e. transitions may manifest *many* states or goals.

[1] These do not state what happens if player and system act at once. We assume a game-tree model; any arbitration scheme works.

Goal Type	**Representation in $A'_{\text{VW}} = \langle L, S, U, I, G \rangle$**
Win	*End* states $S_e \subseteq S$, and $f_{\text{score}} \colon S_e \to \mathbb{N}$ in L
Finish	$S_e \subseteq S$
Prolong	*Non-terminal* states $S_n = S \setminus S_e$
Choose	When $\lvert U_s = \{u \mid u \in U \wedge \text{PRE}(u) \subseteq s\}\rvert > 1$
Configure	*Correct* states $S_c \subseteq S$
Create	$f_{\oplus} \colon C \to 2^C$ in L
Find	$\{\texttt{entity}^1, \texttt{location}^1, \texttt{at}^2\}$ predicates in L
Obtain	$\texttt{has}^2$ predicate in L
Optimize	*Numeric fluents* in L (see Fox and Long 2003)
Reach	$\texttt{adjacent}^2$ predicate in L
Remove	Null constant ϵ and $f_{\ominus} \colon 2^C \to \epsilon$ in L
Solve	When $\lvert U_s \rvert > 1$ results in a state $s \in S_c$
Synchronize	$\psi \leftarrow \forall x, y \exists l \colon \texttt{at}(x, l) \wedge \texttt{at}(y, l)$ and $\mu X.\psi$ holds

Table 3: How the A'_{VW} accommodates the UGO's goals.

As we have demonstrated, goals are central: Theorem 1 observes they are sufficient to induce a game design abstraction. Table 3 summarizes how the A'_{VW} can represent all goal types and linked facets therein, thereby offering evidence of our ontological adequacy. Our future work will formally prove the extent of this coverage.

Epistemological Adequacy for Games Theorem 1 is also the keystone that links models and engines. The critical piece to realizing game system models in a game engine is the *abstraction scheme* $\Upsilon'_{\eth}$ from Eq. 7. Its derivation bridges the abstraction gap for a game system model's implementation, and offers evidence of our epistemological adequacy.

Other game system models are "baked in" to the engines or platforms they are built with, but it is not obvious how one would implement models outside their reference implementation. Our work permits mathematically stating (but does not state for any model but our own in Fig. 1) how a model's operations "ground out" in engines. Our future work will seek implementing other models for comparison.

Conclusion

In this paper, we develop a rigorous link from our novel mathematical foundation of game engines to our novel model of game systems the foundation permits expressing.

Def. 1 is worth revisiting: it represents the lowest level of abstraction one can manipulate in our model. That definition is deliberate—the C'_{VW} (and the A'_{VW} abstraction over it) supports space, time, mechanics, and entities of the Unifying Game Ontology. But for certain games, this is an overcommitment: what of games with no physics? or of Twine games, which lack explicit geometric worlds? And what of games bound by a platform? Future work might explore alternative engine models to answer these questions. At the same time, we offer a (possibly) striking observation: all the

frameworks we cite and the related works *they* cite—more than 15, using different formalisms—rely on *finite state machines* (FSMs); our model relies on FSMs too. Through inductive generalization per enumeration, we conjecture:

Conjecture 1 (Video Game Systems are FSMs). *A finite state machine is a necessary and sufficient model of computation that can characterize all video game systems.*

We leave this conjecture open. However, if an FSM *is* sufficient for game systems, we might revise C'_{VW} to only include time scale $\mathbf{T}$, as it would ground the FSM at the A'_{VW} level. This does not invalidate our work, but begs the question: how representative *is* our chosen C'_{VW}?

In closing, we clarify that we are not interested in formal methods for their own sake. What we model is motivated by game design and game studies concerns; e.g. better understanding game engines' role in shaping the designs of games, how to better conceptualize cross-platform games via engines, etc. We hope to help others build on each other (both *within* research and practice, and *across* it) by being scientifically-grounded and community-relevant.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. #2046294. We also wish to thank Michael Clemens, Justus Robertson, and the anonymous reviewers who were tremendously helpful with their comments during peer review.

References

Aarseth, E.; and Möring, S. 2020. The Game Itself? Towards a Hermeneutics of Computer Games. In *Proc. of the Int'l. Conf. on the Foundations of Digital Games*.

Aarseth, E. J. 1997. *Cybertext: Perspectives on Ergodic Literature*. Baltimore, MD, USA: John Hopkins U. Press.

Abowd, G. D. 1991. *Formal aspects of human-computer interaction*. Ph.D. thesis, U. of Oxford.

Cardona-Rivera, R. E. 2020. Foundations of a Computational Science of Game Design: Abstractions and Tradeoffs. In *Proc. of the 16th AAAI Conf. on AIIDE*, 167–174.

Cardona-Rivera, R. E.; Gardone, M.; Peterson, L.; Hiatt, L. M.; and Roberts, M. 2022. Re-examining the Planning Basis of Goal-driven Autonomy Problems. In *Proc. of the Workshop on Integrated Planning and Execution at ICAPS*.

Debus, M. S. 2019. *Unifying Game Ontology: A Faceted Classification of Game Elements*. Ph.D. thesis, ITU Copenhagen.

Debus, M. S.; Zagal, J. P.; and Cardona-Rivera, R. E. 2020. A Typology of Imperative Game Goals. *Game Studies*, 20(3).

Dormans, J. 2009. Machinations: Elemental Feedback Structues for Game Design. In Saur, J.; and Loper, M., eds., *Proc. of the 5th Int'l. NA Conference on Intelligent Games and Simulation*, 33–40.

Ebner, M.; Levine, J.; Lucas, S. M.; Schaul, T.; Thompson, T.; and Togelius, J. 2013. Towards a Video Game Description Language. In *Artificial and Computational Intelligence in Games*. Dagstuhl.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4): 189–208.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *J. of Artificial Intelligence Research*, 20: 61–124.

García, I.; and Mollál, R. 2005. Videogames decoupled discrete event simulation. *Computers & Graphics*, 29(2): 195–202.

Giambiasi, N.; and Carmona, J. C. 2006. Generalized discrete event abstraction of continuous systems: GDEVS formalism. *Simulation Modelling Practice and Theory*, 14(1): 47–70.

Gregory, J. 2018. *Game engine architecture*. CRC.

Grünvogel, S. M. 2005. Formal models and game design. *Game Studies*, 5(1): 1–9.

Hunicke, R.; LeBlanc, M.; and Zubek, R. 2004. MDA: A Formal Approach to Game Design and Game Research. In *Proc. of the Workshop on Challenges in Game AI*.

Juul, J. 2007. A Certain Level of Abstraction. In *Proc. of the DiGRA Conf.*

Kashima, R.; and Okamoto, K. 2008. General models and completeness of first-order modal μ-calculus. *J. of Logic and Computation*, 18(4): 497–507.

LaValle, S. M. 2019. *Virtual Reality*. Cambridge U. Press.

Lewis, M.; and Jacobson, J. 2002. Game Engines in Scientific Research. *Comm. of the ACM*, 45(1): 27–31.

Lo, P.; Thue, D.; and Carstensdottir, E. 2021. What Is a Game Mechanic? In *Int'l. Conf. on Entertainment Computing*, 336–347.

Martens, C. 2015. Ceptre: A language for modeling generative interactive systems. In *Proc. of the 11th AAAI Conf. on AIIDE*.

Martens, C.; and Hammer, M. A. 2017. Languages of Play: Towards Semantic Foundations for Game Interfaces. In *Proc. of the 12th Int'l. Conf. on the Foundations of Digital Games*, 1–10.

McCarthy, J. 1981. Epistemological problems of artificial intelligence. In *Readings in artificial intelligence*, 459–465. Elsevier.

Nelson, G. 2001. *The Inform Designer's Manual*. Placet Solutions.

Osborn, J. C. 2018. *Operationalizing Operational Logics*. Ph.D. thesis, U. of California, Santa Cruz.

Osborn, J. C.; Lambrigger, B.; and Mateas, M. 2017. HyPED: Modeling and analyzing action games as hybrid systems. In *Proc. of the 13th AAAI Conf. on AIIDE*, 87–93.

Osborn, J. C.; Wardrip-Fruin, N.; and Mateas, M. 2017. Refining operational logics. In *Proc. of the 12th Int'l. Conf. on the Foundations of Digital Games*, 1–10.

Ranzato, F.; and Tapparo, F. 2007. Generalized strong preservation by abstract interpretation. *J. of Logic and Computation*, 17(1): 157–197.

Robertson, J.; and Young, R. M. 2018. Perceptual experience management. *IEEE Transactions on Games*, 11(1): 15–24.

Schaul, T. 2013. A Video Game Description Language for Model-based or Interactive Learning. In *Proc. of the 9th IEEE Conf. on Comp. Inteligence in Games*, 1–8.

Scott, D. 1970. Outline of a mathematical theory of computation. Technical Monograph PRG-2, Oxford University.

Smith, A. M.; Nelson, M. J.; and Mateas, M. 2010. Ludocore: A logical game engine for modeling videogames. In *Proc. of the 6th IEEE Conf. on Comp. Intelligence in Games*, 91–98.

Thue, D.; and Bulitko, V. 2018. Toward a unified understanding of experience management. In *Proc. of the 14th AAAI Conf. on AIIDE*.

Unity Technologies. 2021. Unity Scripting Reference. Version: 2021.3. Last checked: 03-05-22.

Zagal, J. P. 2014. Ontology (in games). In Ryan, M.; Emerson, L.; and Robertson, B., eds., *The Johns Hopkins Guide to the Digital Media*. Johns Hopkins U. Press.

Zagal, J. P.; and Mateas, M. 2010. Time in video games: A survey and analysis. *Simulation & Gaming*, 41(6): 844–868.