УДК 004.942

ВИЗУАЛЬНОЕ МОДЕЛИРОВАНИЕ ИГРОВЫХ МЕХАНИК В ИНСТРУМЕНТЕ TULA: ТЕМПОРАЛЬНЫЕ И ВЕРОЯТНОСТНЫЕ АСПЕКТЫ

В. В. Кугуракова[1] [0000-0002-1552-4910], В. Т. Трофимчук[2] [0009-0001-9106-9614]

[1, 2]*Казанский (Приволжский) федеральный университет, г. Казань, Россия*

[1]vlada.kugurakova@gmail.com, [2]vselord.beta@gmail.com

Аннотация

Рассмотрена проблема визуального представления темпоральных и вероятностных элементов в инструментах формального моделирования игровых механик. На основе анализа существующих систем визуального моделирования – Machinations, сетей Петри, диаграмм состояний UML и UPPAAL – выявлены их ключевые ограничения при описании сложных игровых сценариев. Предложен подход к визуализации темпоральных операторов и вероятностных конструкций в инструменте Tula, разработанном авторами. Описан набор визуальных примитивов, обеспечивающих двунаправленное соответствие между визуальным представлением и формальной спецификацией на языке геймплея. Приведены примеры визуального моделирования типовых игровых сценариев с элементами временных задержек, условных вероятностей и многопользовательских взаимодействий. Показана возможность автоматической трансформации визуальных моделей в формальные спецификации, пригодные для верификации средствами PRISM и UPPAAL. Проведён сравнительный анализ выразительных возможностей Tula и систем-предшественников, показавший, что Tula является единственной из рассмотренных систем, одновременно обеспечивающей формальную верифицируемую семантику, встроенную поддержку темпоральных операторов (X, G, F, U) и произвольных вероятностных распределений, а также визуальную нотацию, организованную вокруг ресурсных потоков и игровых событий. Предложенный в Tula принцип двухуровневой спецификации с двунаправленной трансформацией замыкает цикл «дизайн → формализация → верификация → корректировка», снижая порог входа в формальные методы для практиков без потери верификационной полноты. Это позволяет рассматривать

Tula как промежуточный слой между итеративным геймдизайном и промышленными верификаторами.

***Ключевые слова**: визуальное моделирование, игровые механики, темпоральные операторы, вероятностные модели, формальная верификация, Tula, Machinations, сети Петри.*

ВВЕДЕНИЕ

Формальное описание игровых механик сопряжено с принципиальным противоречием между сложностью математического аппарата и доступностью геймдизайна для практиков. Языки темпоральной логики и вероятностные модели предоставляют необходимую строгость, однако требуют значительной математической подготовки. Визуальные нотации, напротив, снижают порог вхождения, но исторически страдали ограниченной формальной семантикой [1, 2].

В работе [3] нами был предложен язык спецификации геймплея, интегрирующий темпоральные операторы (X, G, F, U) и вероятностные конструкции в единую формальную систему. Настоящая статья посвящена визуальному уровню той же системы: как темпоральные и вероятностные элементы спецификации представляются визуально в инструменте Tula [4] и как это соотносится с подходами систем-предшественников.

Актуальность настоящей работы обусловлена тем, что разрыв между визуальным моделированием и формальной верификацией остается ключевым препятствием на пути к промышленному применению методов верификации в разработке видеоигр. Устранение этого разрыва требует специализированного визуального языка с точно определенной семантикой.

Целью работы было:

1) систематизировать подходы к визуализации темпоральных и вероятностных аспектов в существующих инструментах;

2) описать соответствующие визуальные примитивы в Tula;

3) продемонстрировать на примерах полный цикл от визуальной модели к формальной спецификации и верификации.

ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ ВИЗУАЛЬНОГО МОДЕЛИРОВАНИЯ

Machinations

Система Machinations, предложенная Дормансом [5], является наиболее широко используемым инструментом визуального моделирования игровых механик. Визуальный язык построен на шести типах узлов: Source (источник ресурсов), Pool (хранилище), Drain (потребитель), Converter (преобразователь), Gate (условный маршрутизатор) и Trader (обменник). Ресурсные потоки заданы направленными ребрами с весовыми коэффициентами [5, 6].

С точки зрения временно́го представления Machinations поддерживает дискретный игровой счетчик («time mode»: пошаговый или непрерывный) и узлы с задержкой активации (delayed). Вероятностные аспекты ограничены случайными воротами (random gates) с равномерным распределением. Принципиальным ограничением системы является отсутствие формальной семантики: Machinations-диаграммы не допускают автоматической верификации и не отличают, например, оператор G (всегда) от F (когда-нибудь) [7].

Сети Петри и их расширения

Сети Петри предоставляют математически точную нотацию для моделирования параллельных и асинхронных процессов [8]. В контексте игровых механик они применялись для описания конечных автоматов игровых состояний, систем инвентаря и последовательностей событий. Темпоральные расширения – временны́е сети Петри (Time Petri Nets) и стохастические сети Петри – добавляют соответственно временны́е интервалы на переходах и экспоненциальные распределения задержек [9].

Несмотря на строгость математической основы, стандартные сети Петри обладают существенными ограничениями для геймдизайна: отсутствие иерархии, сложность представления ресурсных потоков с численными значениями, а также низкая интуитивность нотации для не подготовленных математически специалистов [10]. Расширение Coloured Petri Nets частично решает первую и вторую проблемы, однако ценой еще большего усложнения визуального языка [11].

UML-диаграммы состояний

Диаграммы состояний UML (State Machine Diagrams) широко применяются для моделирования логики персонажей и игровых объектов [12]. Они поддерживают иерархические (вложенные) состояния, историю состояний и параллельные регионы. Для игрового искусственного интеллекта (ИИ) диаграммы состояний стали де-факто стандартом благодаря прямой поддержке в движках Unity и Unreal Engine.

Темпоральные аспекты в UML-диаграммах выражаются через временны́е события («after(t)» на переходе) и диаграммы времени (Timing Diagrams). Однако вероятностные переходы в стандарте UML не предусмотрены: их реализация требует нестандартных расширений или перехода к вероятностным автоматам [13]. Верификация UML-моделей возможна через конвертацию в формат UPPAAL[1] [14] или PRISM[2] [15], но инструментальная поддержка этого пути остается ограниченной.

Среда моделирования UPPAAL

UPPAAL [14] – верификатор временны́х автоматов (timed automata) – предоставляет собственный визуальный редактор сетей автоматов. Каждый автомат описывается графом состояний с инвариантами на состояниях, а также охранными условиями (guards) и сбросами часов (resets) на переходах. Язык запросов TCTL[3] [16] позволяет задавать темпоральные свойства: достижимость (EF), безопасность (AG), жизнеспособность (AF).

Для игровых применений UPPAAL использовался в задачах верификации балансировки [3] и анализа инвариантов игрового мира. Ограничением UPPAAL как проектировочного инструмента является его ориентация на верификацию, а не на синтез: модели создаются «снизу вверх» от формальных конструкций, что затрудняет применение в итеративном геймдизайне.

[1] UPPAAL – верификатор систем реального времени на основе сетей временны́х автоматов (Упсальский университет / Университет Ольборга).

[2] PRISM (Probabilistic Symbolic Model Checker) – верификатор вероятностных систем, поддерживающий DTMC, MDP и логики PCTL/CSL (Университет Бирмингема).

[3] TCTL (Timed Computation Tree Logic) — расширение логики ветвящегося времени CTL, добавляющее количественные временны́е ограничения на операторы достижимости и безопасности вида EF≤t, AG≥t. Используется как язык запросов в UPPAAL [14].

Сравнительный анализ

Сводная информация о рассмотренных системах представлена в табл. 1. Установлено принципиальное противоречие между двумя группами систем. Machinations и частично UML-диаграммы состояний оптимизированы для практики геймдизайна: их визуальные нотации интуитивны, нотации ориентированы на ресурсные потоки и поведение игровых объектов. Однако обе системы жертвуют формальной строгостью: в Machinations отсутствует верифицируемая семантика, а UML предоставляет ее лишь через нестандартные расширения. Сети Петри и UPPAAL, напротив, обладают строгой математической основой и допускают автоматическую верификацию, однако их визуальные языки не адаптированы к специфике игровых задач: ресурсные потоки, балансировка и итеративное прототипирование находятся за пределами их проектной области.

Табл. 1. Характеристики систем визуального моделирования игровых механик.

Критерий	Machinations	Сети Петри	UML SM	UPPAAL	Tula
Формальная семантика	Нет	Да	Частично	Да	Да
Временны́е операторы	Ограничено	Расширения	after(t)	Полные	Полные
Вероятностные элементы	Случайные ворота	Стохастические расширения	Нет	Нет*	Встроены
Ресурсные потоки	Да	Ограничено	Нет	Нет	Да
Автоверификация	Нет	Да	Через конвертацию	Да	Да
Ориентация на геймдизайн	Да	Нет	Частично	Нет	Да

*вероятностная верификация выносится в PRISM (либо UPPAAL-SMC).

Особого внимания заслуживает различие в подходах к темпоральным элементам. Machinations использует неявную темпоральность: время моделируется счетчиком итераций симуляции без прямого соответствия операторам формальной логики. Временны́е сети Петри и UPPAAL реализуют непрерывное или дискретное время с точной семантикой, однако их конструкции (инварианты состояний, часовые переменные) не имеют интуитивных аналогов в лексиконе

геймдизайна. UML-диаграммы занимают промежуточную позицию: конструкция after(t) понятна практикам, но ее выразительной мощности недостаточно для описания свойств вида «с вероятностью не менее 0.9 игра завершается за 20 ходов».

Вероятностное измерение обнаруживает схожую картину. Случайные ворота Machinations обеспечивают базовое моделирование недетерминизма, однако ограничены равномерным распределением и не допускают задания условных вероятностей.

Стохастические сети Петри поддерживают экспоненциальные распределения, что адекватно для телекоммуникационных задач, но неестественно для игровой механики, оперирующей дискретными событиями с произвольными вероятностями. PRISM как верификатор марковских цепей обладает наибольшей выразительностью в вероятностном домене, однако является инструментом верификации, а не проектирования.

Наглядное сравнение нотаций на примере одной механики представлено на рис. 1 и 2. Нотация Machinations (рис. 1) обеспечивает интуитивное визуальное прототипирование игровых экономик. Вместе с тем в базовой версии инструмента отсутствуют темпоральные метки; вероятности в Random Gate фиксированы и не зависят от игрового состояния; формальная семантика не определена, что исключает верификацию без привлечения расширений типа Micro-Machinations [17].

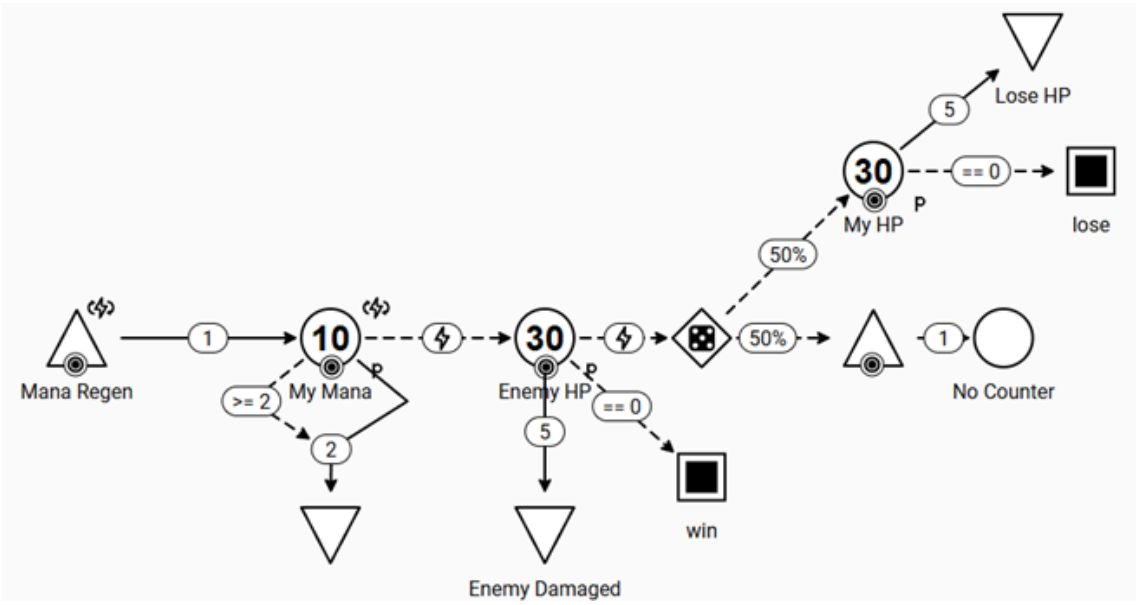

Рис. 1. Диаграмма Machinations для боевой механики карточной игры.

В отличие от Machinations, нотация временны́х сетей Петри (рис. 2) допускает строгую формальную трактовку, однако применительно к задачам геймдизайна выявляется иной набор ограничений: ресурсные потоки выражаются кратностью дуг без числовых значений; стохастические переходы задаются интенсивностью λ (экспоненциальное распределение в рамках GSPN), что не позволяет напрямую задать дискретные вероятности вида $p = 0.3/0.7$.

Таким образом, ни одна из рассмотренных систем не обеспечивает одновременно: (1) формальной семантики, допускающей верификацию; (2) поддержки темпоральных и произвольных вероятностных конструкций; (3) визуального языка, ориентированного на понятия игровых механик.

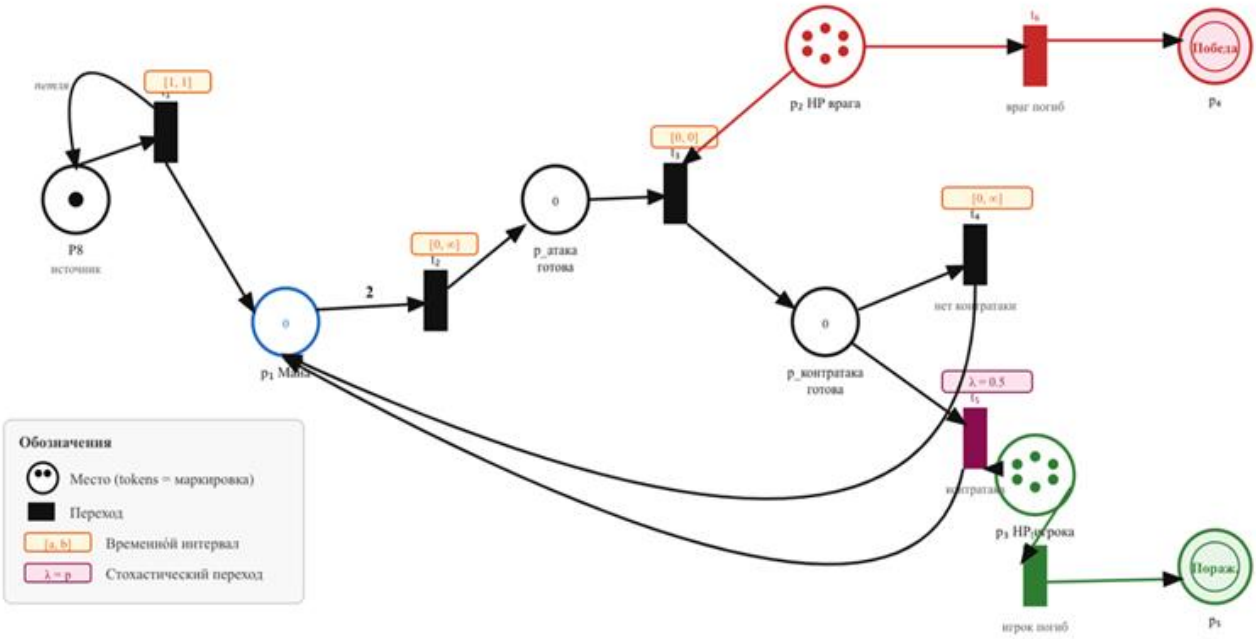

Рис. 2. Временна́я сеть Петри для той же механики.

Именно эта совокупность требований определила архитектурные решения инструмента Tula, описанные в следующем разделе.

ВИЗУАЛЬНЫЙ ЯЗЫК ИНСТРУМЕНТА TULA

Архитектура двухуровневой спецификации

Ключевым архитектурным решением Tula является принцип двухуровневой спецификации: высокоуровневая (формальная) и низкоуровневая (визуаль-

ная) спецификации существуют как два согласованных представления одной модели. Высокоуровневая спецификация на языке геймплея содержит полное формальное описание, включая темпоральные и вероятностные операторы, предназначенное для верификации [18], низкоуровневая визуальная спецификация – упрощенное представление для итеративного прототипирования и балансировки [4].

Соответствие между уровнями задано набором правил трансформации, позволяющих как порождать формальную спецификацию из визуальной модели, так и восстанавливать визуальное представление по существующей спецификации. Это обеспечивает непротиворечивость обоих уровней в процессе совместного редактирования.

Базовые визуальные примитивы

Визуальный язык Tula включает восемь базовых типов узлов (см. рис. 3), образующих минимально достаточный набор для описания игровых механик.

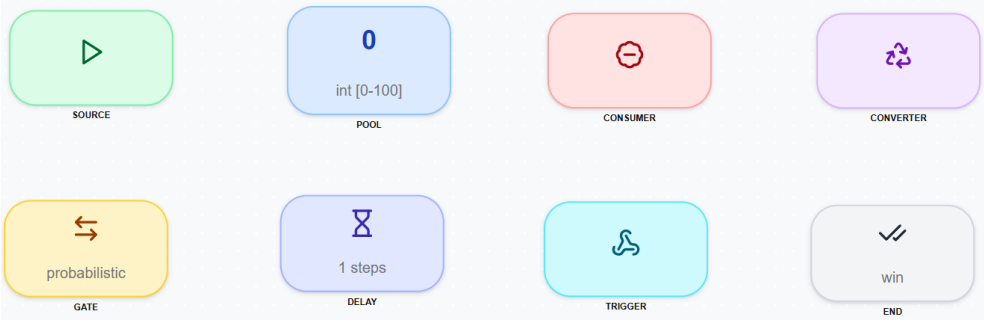

Рис. 3. Визуальные примитивы Tula.

Узел *Source* (Источник) порождает ресурсы по заданному правилу активации. В отличие от Machinations, Tula позволяет задавать произвольное распределение интенсивности генерации, включая непрерывные распределения (нормальное, показательное) и темпорально обусловленные правила: «генерировать каждые *t* ходов, начиная с хода *n*».

Узел *Pool* (Хранилище) хранит ресурсы с заданными ограничениями емкости. Каждый Pool несет тип ресурса, текущее значение и диапазон допустимых значений [*min*, *max*], что соответствует формальному определению состояния $s \in S$ с ограничением $s.value \in [s.min, s.max]$.

Узел *Consumer* (Потребитель) необратимо поглощает ресурсы. В контексте игровой механики он соответствует «стоку», например расходу маны при применении заклинания или потере здоровья при получении урона.

Узел *Converter* (Конвертер) преобразует один тип ресурса в другой по заданному коэффициенту конвертации. Конвертер поддерживает как детерминированные, так и вероятностные коэффициенты: например, «с вероятностью 0.3 конвертировать 2 единицы атаки в 3 единицы урона».

Узел *Gate* (Условный маршрутизатор) направляет ресурсный поток по одному из нескольких исходящих ребер в зависимости от булева или вероятностного условий. Узел помечается как probabilistic (вероятностный), если нужно отправление ресурсов по нескольким рёбрам согласно указанной вероятности, или deterministic (булев), когда выбирается один путь из многих. Именно через *Gate* визуализируются вероятностные развилки и условные переходы формальной спецификации.

Узел *Delay* (Задержка) является одним из двух специализированных примитивов для темпоральных конструкций. Он буферизует проходящий ресурсный поток на заданное количество временны́х шагов δ, после чего передает его дальше. Формально это соответствует применению оператора $X \wedge \delta$ (следующий на δ шагов) к соответствующему событию.

Узлы *Trigger* и *End* обеспечивают соответственно принудительную активацию связанных узлов при наступлении события и фиксацию конечных состояний игры (победу, поражение, ничью).

Типы ребер и их семантика

Наряду с узлами семантически нагруженными являются и типы соединительных ребер. Tula поддерживает четыре типа ребер: прямое ресурсное ребро (непрерывный поток ресурсов), триггерное ребро (передача управляющего сигнала без ресурса), ребро-модификатор (изменение параметра целевого узла) и ребро обратной связи (установление зависимости интенсивности потока от состояния целевого узла). Последний тип особенно важен для моделирования динамики балансировки [19] и не имеет прямого аналога в базовом Machinations.

ВИЗУАЛИЗАЦИЯ ТЕМПОРАЛЬНЫХ ЭЛЕМЕНТОВ

Оператор «Следующий» (X)

Темпоральный оператор Xφ («в следующий момент времени φ») визуализируется в Tula через узел *Delay* с параметром $\delta = 1$ (см. рис. 4).

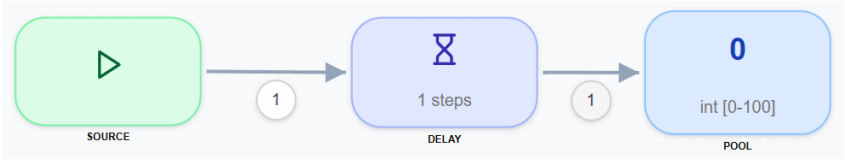

Рис. 4. Три узла этого примера отражают передачу (СЛЕДУЮЩИЙ МОМЕНТ) потока ресурсов из источника в пул с использованием временного буфера.

Если формальная спецификация содержит правило вида (см. лист. 1), то в визуальной модели это представляется как ребро от *Source* «Атака» через узел *Delay(1)* к *Pool* «Здоровье противника».

Листинг 1. Пример события, наступающего в будущем.

```
Event AttackPlayer {
    Effect: X(Target.Health -= Self.Attack)
}
```

Наглядность такого представления обусловлена явной «физической» интерпретацией задержки как буфера ресурса во времени.

Оператор «Всегда» (G) и инварианты

Оператор Gφ («всегда выполняется φ») соответствует инварианту системы. В визуальной нотации Tula инварианты представляются как постоянно активные ограничения на значения *Pool*-узлов (через параметры min/max) или через петли обратной связи, поддерживающие значение ресурса в допустимом диапазоне (см. рис. 5). Двойным нажатием открываются опции ноды Pool. В них можно указать min и max значения, что соответствует G(>Min) и G(≤Max).

Рис. 5. Пул со встроенными инвариантами (то есть то, что выполняется ВСЕГДА).

Например, инвариант «здоровье игрока всегда неотрицательно» (G(*Player.Health* ≥ 0)) задается ограничением *Pool.min* = 0 на соответствующем узле.

Оператор «Когда-нибудь» (F) и цели

Оператор Fφ («когда-нибудь φ») описывает достижимые состояния и игровые цели. В Tula он визуализируется через узел *End*, связанный условным ребром с проверяемым *Pool*-узлом (см. рис. 6). Когда значение в *Pool* достигает заданного порога, активируется *End*, фиксируя наступление целевого состояния. Таким образом, условие победы «игрок набирает 1000 очков» (F(*Score* ≥ 1000)) непосредственно кодируется топологией схемы.

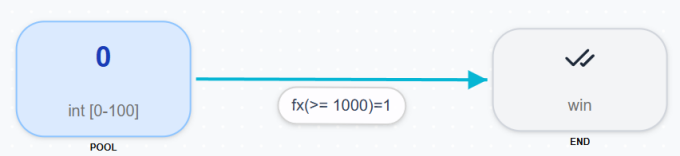

Рис. 6. Пример реализации функции КОГДА-НИБУДЬ.

Оператор «До» (U) и ограниченные свойства

Оператор ϕUψ («ϕ выполняется до тех пор, пока не выполнится ψ») является наиболее сложным для визуального представления. В Tula он реализуется с помощью двух узлов – *Trigger* и *Pool,* которые совместно моделируют выполнение φ.

Ключевым элементом является условная обратная связь от узла *Pool* к самому себе: на этом ребре задано условие «не ψ». Пока это условие истинно (т. е. ψ еще не наступило), цикл активен и φ выполняется непрерывно. Узлы *Trigger* и *Pool* соединены прямой связью, которая обеспечивает переход к завершению оператора в момент, когда ψ становится истинным (условие на обратной связи перестает выполняться). Данная конструкция визуально сложнее одиночных операторов, что отражает реальную сложность формального понятия «пока не» (см. рис. 7).

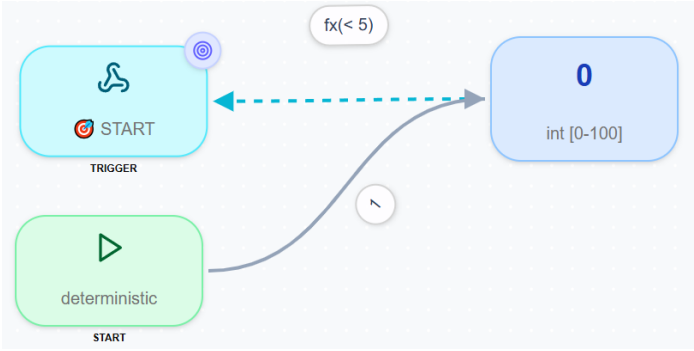

Рис. 7. Детерминированный узел Start подаёт ресурс в счётчик; цикл активен, пока не выполнится условие (в примере — пока счётчик не станет равен 5). Свойство цели триггера указывает, куда возвращается поток (на узел Start); при выполнении условия триггер разрывает цикл.

Ограниченные по времени свойства

Для ограниченных временны́х свойств вида F[0, *T*]*φ* (достижимость в пределах *T* шагов) Tula вводит специализированный визуальный элемент – временно́й счетчик (*Timer*). Это узел, автоматически инкрементирующий свое значение на каждом временно́м шаге и связанный с *Gate*, закрывающим поток при превышении лимита *T*. Сравнение с UPPAAL здесь особенно уместно: часовые переменные UPPAAL выполняют аналогичную функцию, но в контексте автоматной модели, а не ресурсного потока. Соответствие всех четырех операторов сведено в табл. 2.

Табл. 2. Соответствие темпоральных операторов визуальным конструкциям.

Свойства	СЛЕДУЮЩИЙ	ВСЕГДА	КОГДА-НИБУДЬ	ДО
Темпоральный оператор	X*φ*	G*φ*	F*φ*	*φ*U*ψ*
Формальная запись	*Effect: X*(*HP*– = *Atk*)	*G*(*Health* > 0), *G*(*Mana* ≤ *MaxMana*)	*F*(*Score* ≥ 1000), *F*(*HP* ≤ 0)	*ShieldActive U EnemyDefeated*
Узел	*Delay*(*δ* = 1)	*Pool* с ограничениями min/max	*End*-узел с условием на ребре.	Узлы *Trigger* и *Pool*, соединенные обратной условной связью, на ребре указано условие выполнение цикла.
Семантика	*Delay*(*δ*) буферизует поток на *δ*	Инвариант *G* кодируется	*End* активируется, когда условие на ребре →	Цикл моделирует «*φ* всегда». Ребро

	шагов, затем передает далее. Соответствует X ^ *n* *φ* (*n* = *δ*)	ограничениями min/max на *Pool*. Нарушение невозможно структурно.	true. Пометка *End* как «измеримого» автогенерирует PRISM-запрос.	обратной связи отслеживает *ψ* = false; при *ψ* = true триггер разрывает цикл.
Аналог в верификаторе	Сброс часовой переменной в UPPAAL.	Инвариант состояния в UPPAAL / NuSMV	PRISM: *P* = ? [F *Score* ≥ 1000] – автогенерация запроса.	Наиболее сложная конструкция – отражает сложность оператора U.
Пример	Урон наносится не сразу, а на следующем ходе.	Здоровье не станет отрицательным.	Победа при достижении 1000 очков.	Щит активен, пока враг не побежден.
Схема	см. рис. 4	см. рис. 5	см. рис. 6	см. рис. 7

ВИЗУАЛИЗАЦИЯ ВЕРОЯТНОСТНЫХ ЭЛЕМЕНТОВ

Вероятностные ворота

Основным примитивом для вероятностных конструкций является *Gate* в вероятностном режиме. В отличие от Machinations, где вероятностные ворота поддерживают только равномерное распределение по выходам, Tula позволяет задавать произвольные дискретные распределения с явными метками вероятностей на исходящих ребрах. Например, правило атаки с 50-процентным шансом получить ответный урон (см. лист. 2) визуализируется как *Gate* «Контратака?» с двумя исходящими ребрами: «Да (0.5)» → *Consumer* здоровья атакующего и «Нет (0.5)» → *End* (нет эффекта).

Листинг 2. Пример вероятностного события (контратака с вероятностью 0.5).

```
Event AttackCard {
    Target: Card
    Effect: Target.Health -= Self.Attack
    P[Self.Health -= Target.Attack] = 0.5
}
```

Метки вероятностей на ребрах являются обязательным элементом нотации и проверяются при валидации: сумма вероятностей на всех исходящих ребрах из *Gate* в вероятностном режиме должна равняться 1. Сравнение вероятностного ветвления *Gate* в Machinations и Tula приведено на рис. 8 и 9.

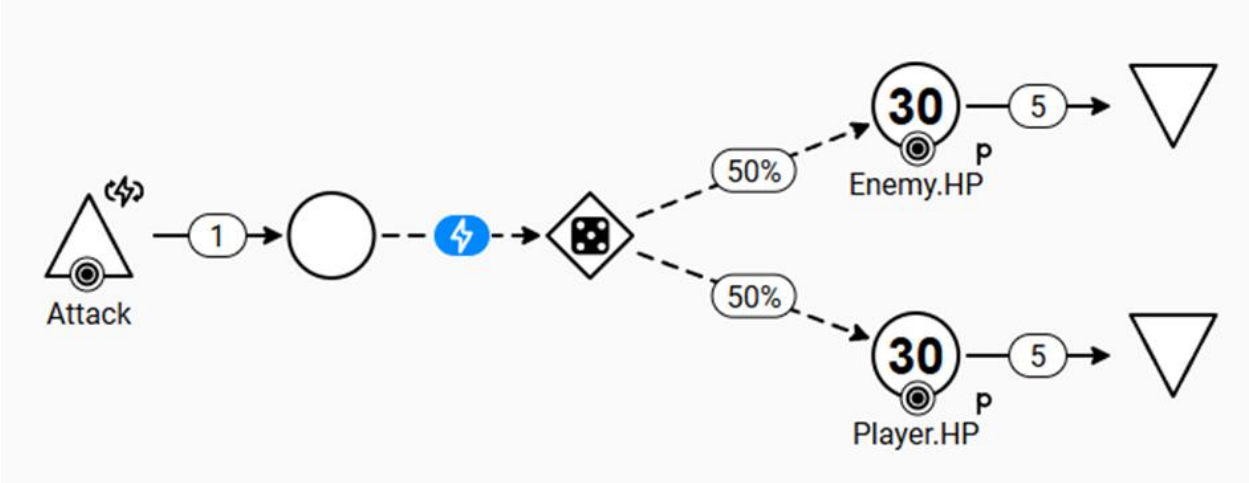

Рис. 8. Вероятностный *Gate* в Machinations.

В Machinations (рис. 8) *Random Gate* распределяет ресурс равномерно: веса выходов фиксированы и не могут зависеть от состояния игры, условные вероятности вида $P[\varphi \mid \psi]$ недоступны, верификационный запрос $P = ?\ [F\varphi]$ не формируется.

В Tula (рис. 9) узел *Gate* в вероятностном режиме допускает произвольные дискретные вероятности p_n при условии $\Sigma p_n = 1$, которое проверяется автоматически; значения p_n могут изменяться через ребро-модификатор, связанное с Monitor-узлом, отслеживающим текущее состояние игры.

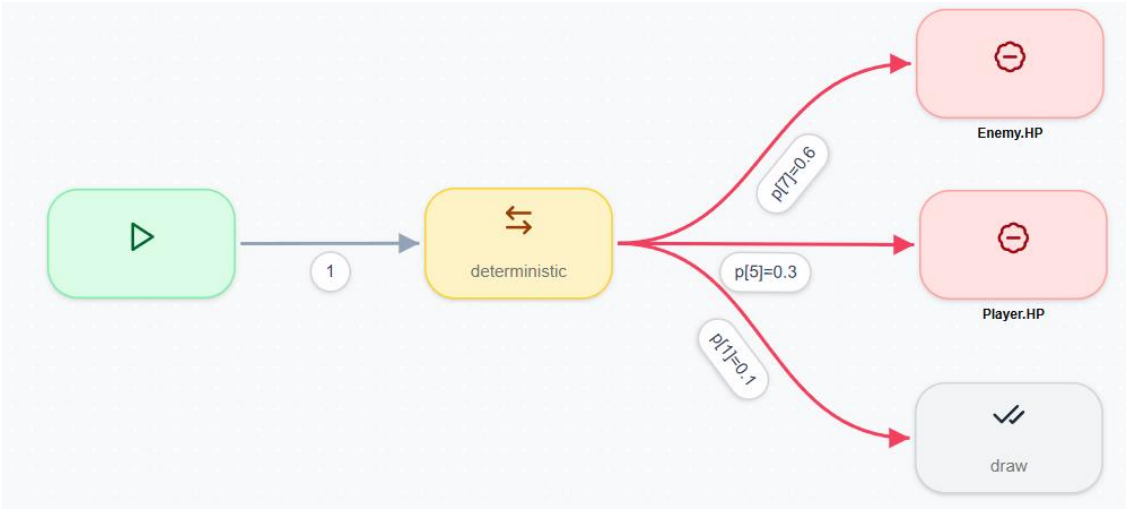

Рис. 9. Вероятностный *Gate* в Tula. На приведенном примере вероятность передачи одного ресурса (p[1]) 10%, суммарно все рёбра, выходящие из Gate должны давать 100% (при невыполнении будет выдано предупреждение).

Таким образом, вероятностная модель Tula охватывает класс механик, который нельзя получить средствами базовой нотации Machinations.

Условные вероятности

Условные вероятности вида $P[\varphi|\psi] = p$ («вероятность φ при условии ψ») представляются цепочкой из двух *Gate*-узлов. Первый (в булевом режиме) проверяет условие ψ, второй (в вероятностном режиме, активируемый только при ψ = true) реализует распределение вероятностей событий при выполненном условии. Такая двухступенчатая структура соответствует формальному определению условной вероятности и сохраняет читаемость схемы.

Вероятностные свойства баланса

Особый класс составляют свойства вида $P = ? [F\varphi]$ – вычисление вероятности достижения целевого состояния. Tula не проводит вероятностную верификацию напрямую (для этого используется экспорт в PRISM [15], реализующий логику PCTL [20]), однако обеспечивает визуальный интерфейс для задания вопросов верификации. Пользователь отмечает целевой *End*-узел как «измеряемый», и инструмент автоматически формирует соответствующий PRISM-запрос $P = ?[F$ *target_state*] и передает его верификатору.

Марковские цепи и стохастические стратегии

Для моделирования стохастических стратегий, где вероятности переходов зависят от истории игры, Tula вводит понятие контекстно зависимого *Gate*. Такой узел обращается к значениям нескольких *Pool*-узлов при вычислении выходных вероятностей, реализуя тем самым переходную функцию марковской цепи с частичной наблюдаемостью [21]. Визуально это представляется входными ребрами-модификаторами от информационных *Pool*-узлов к *Gate* – ребрами, изображаемыми пунктирной линией для отличия от ресурсных ребер.

ПРИМЕР: ВИЗУАЛЬНАЯ МОДЕЛЬ КАРТОЧНОЙ ИГРЫ

Описание сценария

Под игровой механикой понимаются правило или система правил, определяющие допустимые действия игрока и их последствия в игровом мире [22].

В качестве сквозного примера используется карточная игра со стихийной системой, описанная нами в работе [3]. Игра включает двух игроков, колоды карт четырех стихий (Огонь, Вода, Земля, Воздух), систему маны с инкрементом per turn и боевую механику с элементальными преимуществами. Этот пример выбран как достаточно сложный, чтобы продемонстрировать все ключевые визуальные конструкции Tula, и достаточно известный, чтобы результаты были интуитивно понятны.

Визуальная структура системы маны

Система маны реализует инвариант «количество маны не превышает максимум» – G(*Mana* ≤ *MaxMana*) и темпоральное правило инкремента «в начале каждого хода» – G(X(*Mana* = min(*Mana* + 1*, MaxMana*))). Визуально это задается следующей конструкцией: *Source* «Прирост маны» с активацией *Trigger.OnTurnStart* направляет 1 единицу ресурса «Мана» в *Pool* «Мана Игрока» [0, 10]. Ограничение max = 10 на *Pool* автоматически отсекает избыток, реализуя оператор min.

Визуальная структура боевой механики

Атака карты включает как детерминированный компонент (нанесение урона цели), так и вероятностный (ответный урон с вероятностью 0.5) и темпоральный (элементальные преимущества, вычисляемые перед нанесением урона). В Tula это представляется цепочкой: *Trigger* «Атака» → *Converter* «Вычисление стихийного модификатора» → *Gate* «Тип атаки?» (детерминированная / с ответным уроном, *p* = 0.5) → ветви к соответствующим *Consumer*-узлам «Здоровье цели» и «Здоровье атакующего».

Converter «Вычисление стихийного модификатора» реализует условную логику элементальных преимуществ. Его выходной коэффициент задается формулой с обращением к *Pool*-узлам «Стихия атакующего» и «Стихия цели»; численно это ×1.5 при Огонь → Воздух и ×1.2 при Огонь → Игрок. Данная конструкция демонстрирует выразительные возможности *Converter* в Tula: в отличие от Machinations, где конвертер имеет фиксированный коэффициент, Tula допускает вычисляемые зависимые коэффициенты.

Верификационные запросы

Из визуальной модели автоматически генерируются, например, следующие верификационные запросы (см. рис. 10 и 11).

Для PRISM:

- `P=? [F Player1.Health<=0]` – вероятность поражения P1;
- `P=? [G<=20 (HP1>0&HP2>0)]` – вероятность затяжной игры в 20 ходов;
- `P=? [F<=30 (EnemyHP<=0)]` – вероятность победы не позднее 30-го хода;
- `R{"turns"}=? [F(EnemyHP<=0)]` – ожидаемое число ходов до победы.

Для UPPAAL:

- `A<>GameOver` – проверка завершимости;
- `A[](Player1Turn -> Player2Turn)` – корректность чередования ходов.

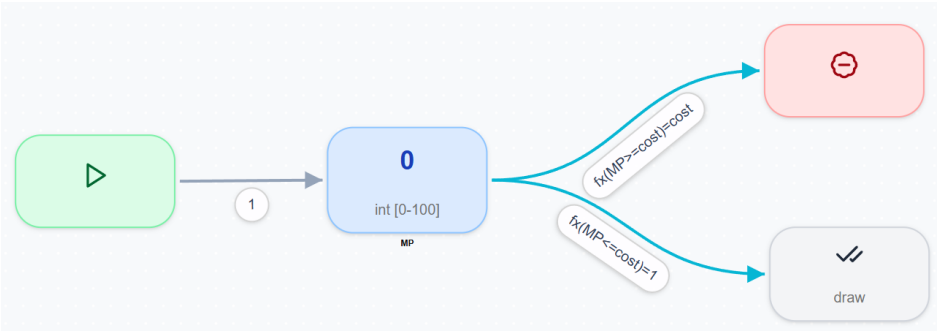

Рис. 10. Система управления ресурсом маны в нотации Tula; инвариант $G(Mana \le MaxMana) \wedge G(Mana \ge 0)$ формулируется автоматически по структуре *Pool*-узла.

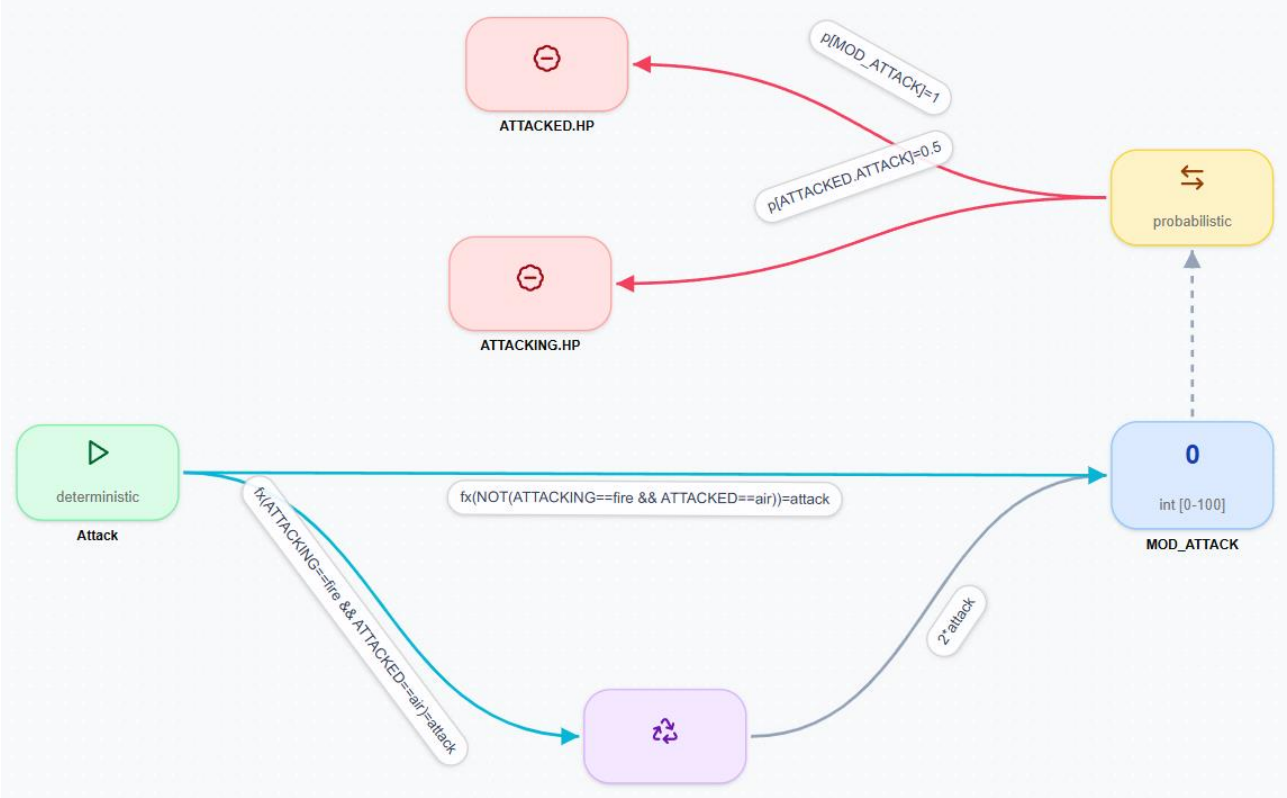

Рис. 11. Полная модель боевой механики карточной игры в нотации Tula; по структуре модели автоматически генерируются инварианты *G* для Pool-узла модифицированной атаки и верификационные запросы для PRISM и UPPAAL.

Подобное прямое соответствие визуальных элементов верификационным запросам является ключевым преимуществом Tula: геймдизайнер, маркируя узлы и ребра при проектировании, одновременно формулирует проверяемые свойства системы.

ТРАНСФОРМАЦИЯ ВИЗУАЛЬНОЙ МОДЕЛИ В ФОРМАЛЬНУЮ СПЕЦИФИКАЦИЮ

Алгоритм трансформации

Трансформация визуальной модели в формальную спецификацию осуществляется по систематической схеме: каждому типу узла соответствует конкретная конструкция языка геймплея. Функция трансформации определяется правилами, приведенными в табл. 3.

Табл. 3. Правила трансформации узлов Tula в конструкции формальной спецификации.

Узел Tula	Конструкция спецификации	Пример
Pool	`State { Name : Type[min,max] = init }`	`HP: Int[0,30]=30`
Source	`Event { Trigger; Effect }`	`AttackCard`
Gate (булев)	`Rule { Condition→Effect }`	`if Mana>=cost`
Gate (вероятностный)	`P[Effect]=p`	`P[X(HP-=atk)]=0.5`
Delay(δ)	`X ^ δ(Effect)`	`X^1(Self.HP-=atk)`

Пример применения правил трансформации для фрагмента боевой механики показан на рис. 12: исходный *Pool*-узел с параметрами [0, 30] преобразуется в объявление состояния, а *Source*-узел с вероятностным *Gate* и *Delay*-узлом – в событие с темпоральным вероятностным эффектом.

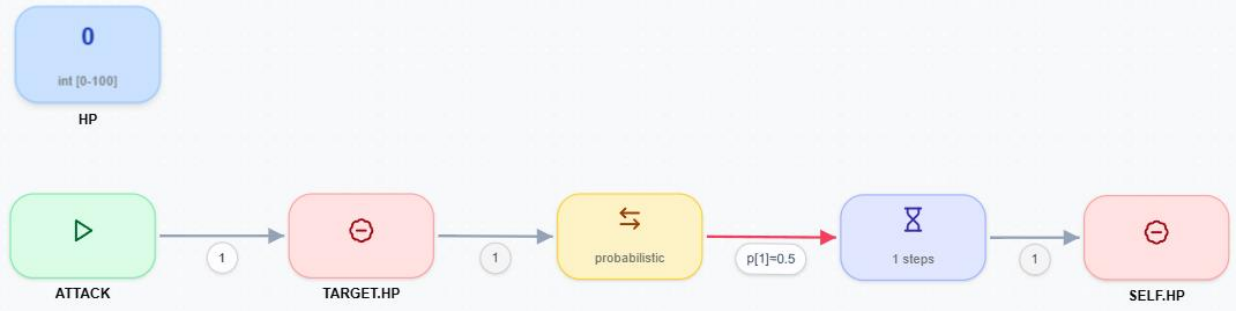

Рис. 12. Фрагмент визуальной модели боевой механики в Tula.

Трансформации узлов остальных типов строятся по той же схеме в соответствии с табл. 3. Фрагмент формальной спецификации (см. лист. 3) получен применением данных правил из визуальной модели (рис. 12).

Листинг 3. Формальная спецификация визуальной модели.

```
// Трансф. Pool → State
State {
   HP: Int[0, 30] = 30
}
// Трансф. Source + Gate + Delay
Event AttackCard {
   Target: Card
   Effect: Target.HP -= Self.Attack
   P[X(Self.HP -= Target.Attack)]   X'
   = 0.5                            Delay(1)
}
```

Гарантии корректности

Алгоритм трансформации обеспечивает ряд гарантий корректности.

Сохранение ресурсов: для каждого типа ресурса суммарный поток из *Sources* равен суммарному потоку в *Consumers* и *Pools*, что соответствует инварианту сохранения в формальной модели.

Полнота: каждый узел визуальной модели трансформируется ровно в одну конструкцию спецификации; неоднозначностей нет.

Детерминизм: для детерминированных переходов (*Gate* в булевом режиме) формальная спецификация не содержит вероятностных операторов; вероятностные операторы появляются только там, где визуальная модель содержит вероятностные ворота.

Ограничения

Алгоритм трансформации имеет и ограничения. Некоторые сложные темпоральные свойства, выразимые в TCTL, не имеют прямого визуального аналога в текущей версии Tula: в частности, вложенные операторы Until и свойства с альтернирующей квантификацией. Для их задания пользователь может вручную дополнять автоматически сгенерированную спецификацию. Данное ограничение является предметом дальнейшего развития инструмента.

ОБСУЖДЕНИЕ

Предложенный подход к визуализации темпоральных и вероятностных элементов в Tula позволяет устранить ключевые ограничения систем-предшественников. По сравнению с Machinations, Tula обеспечивает формальную семантику визуальных элементов, допускающую автоматическую верификацию. В отличие от сетей Петри и UPPAAL, Tula сохраняет ориентацию на задачи геймдизайна: визуальный язык организован вокруг понятий ресурсов, потоков и игровых событий, а не математических абстракций.

Важным преимуществом такого подхода является принцип двунаправленного соответствия: изменения в визуальной модели немедленно отражаются в формальной спецификации и наоборот. Это поддерживает итеративный процесс разработки, при котором геймдизайнер может работать на визуальном уровне, а инженер по верификации – на уровне формальных запросов, без рассинхронизации представлений.

Сравнение с UPPAAL показало, что Tula не претендует на замену специализированных верификаторов. Вместо этого Tula выступает промежуточным слоем — визуальным интерфейсом — к формальным методам, снижающим барьер входа для геймдизайнеров при сохранении полноты верификационных возможностей через экспорт в PRISM и UPPAAL.

Среди ограничений следует отметить: неполное покрытие операторов временно́й логики (в особенности вложенных Until-конструкций), отсутствие поддержки непрерывного времени (Tula работает в дискретном временно́м домене) и текущее ограничение на число узлов в модели с приемлемым временем верификации (порядка 50–100 узлов для задач балансировки).

ЗАКЛЮЧЕНИЕ

Рассмотрен подход к визуальному представлению темпоральных и вероятностных элементов игровых механик в инструменте Tula. Проведенный анализ систем-предшественников (Machinations, сети Петри, UML-диаграммы состояний, UPPAAL) показал, что ни одна из них не обеспечивает одновременно формальной семантики, поддержки вероятностных элементов и ориентации на задачи геймдизайна.

Предложенный набор визуальных примитивов – расширенные узлы *Delay*, *Timer*, контекстнозависимый *Gate* и принцип двухуровневой спецификации – позволяет представлять темпоральные операторы X, G, F, U [23] и вероятностные конструкции непосредственно в визуальной нотации с гарантированным соответствием формальной спецификации. Автоматическая трансформация визуальной модели в спецификацию на языке геймплея, пригодную для верификации средствами PRISM и UPPAAL, замыкает цикл «дизайн → формализация → верификация → корректировка».

Таким образом, Tula не подменяет промышленные верификаторы, а служит промежуточным слоем между итеративным геймдизайном и формальной верификацией, снижая порог входа в формальные методы для практиков без потери верификационной полноты.

Направления дальнейшего развития включают расширение набора поддерживаемых темпоральных конструкций, разработку визуальной нотации для стохастических игровых стратегий и интеграцию прямой верификации внутри Tula без экспорта во внешние инструменты.

Благодарности

Работа выполнена за счет гранта, предоставленного Академией наук Республики Татарстан образовательным организациям высшего образования, научным и иным организациям на поддержку планов развития кадрового потенциала в части стимулирования их научных и научно-педагогических работников к защите докторских диссертаций и выполнению научно-исследовательских работ (Соглашение от 22.12.2025 № 12/2025-ПД-КФУ)

СПИСОК ЛИТЕРАТУРЫ

1. *Adams E., Dormans J.* Game Mechanics: Advanced Game Design. Berkeley: New Riders, 2012. 360 p.

2. *Dormans J.* Engineering Emergence: Applied Theory for Game Design: dis. … dr. sci. Amsterdam: University of Amsterdam, 2012. 328 p.

3. *Кугуракова В.В.* Интеграция темпоральных и вероятностных элементов в язык спецификации игрового процесса // Машиностроение: сетевой электронный научный журнал, 2026 (принято к публикации).

4. *Рахманкулова В.Р., Кугуракова В.В.* Онлайн-инструмент Tula для балансировки видеоигр // Электронные библиотеки. 2025. Т. 28, № 4. С. 903–930.

5. *Dormans J.* Machinations: Elemental feedback patterns for game design // Proceedings of the 5th International North American Conference on Intelligent Games and Simulation (EUROSIS). 2009. P. 33–40.

6. *Dormans J.* Simulating mechanics to study emergence in games // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 2011. Vol. 7, No. 3. P. 2–7.

7. *Grünvogel S.M.* Formal Models and Game Design // Game Studies. 2005. Vol. 5, No. 1. URL: http://www.gamestudies.org/0501/gruenvogel/

8. *Peterson J.L.* Petri Net Theory and the Modeling of Systems. Prentice Hall, 1981. 290 p.

9. *Molly M.K.* Performance Analysis Using Stochastic Petri Nets // IEEE Trans. Comput. 1982. Vol. 31, No. 9. P. 913–917.

10. *Martens C.* Generative Structural Analysis of Petri Nets // Trans. on Petri Nets and Other Models of Concurrency XI. 2016. P. 37–56.

11. *Jensen K.* Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1. Berlin: Springer, 1997. 234 p.

12. About the Unified Modeling Language Specification Version 2.5.1 // OMG. OMG Unified Modeling Language (OMG UML) [Электронный ресурс]. URL: https://www.omg.org/spec/UML/2.5.1 (дата обращения: 16.02.26)

13. *Segala R.* Modeling and Verification of Randomized Distributed Real-Time Systems: dis. … dr. sci. MIT, 1995.

14. *Larsen K.G., Pettersson P., Yi W.* UPPAAL in a Nutshell // Int. J. Software Tools for Technology Transfer. 1997. Vol. 1, No. 1. P. 134–152.

15. *Kwiatkowska M., Norman G., Parker D.* PRISM 4.0: Verification of Probabilistic Real-Time Systems // Computer Aided Verification. 2011. P. 585–591.

16. *Alur R., Courcoubetis C., Dill D.* Model-Checking in Dense Real-Time // Information and Computation. 1993. Vol. 104, No. 1. P. 2–34.

17. *Klint P., van Rozen R.* Micro-Machinations: A DSL for Game Economies // Software Language Engineering: 6th Int. Conf., SLE 2013, Indianapolis. 2013. Vol. 8225. P. 36–55.

18. *Кугуракова В.В.* Формальный подход к пространственно-временному моделированию игровых систем // Ученые записки Казанского университета. Серия: Физико-математические науки. 2024. Т. 166, № 4. С. 532–554.

19. *Сахибгареева Г.Ф., Кугуракова В.В., Большаков Э.С.* Инструменты балансирования игр // Электронные библиотеки. 2023. Т. 26, № 2. С. 225–251.

20. *Hansson H., Jonsson B.* A Logic for Reasoning About Time and Reliability // Formal Aspects of Computing. 1994. Vol. 6, No. 5. P. 512–535.

21. *Kaelbling L.P., Littman M.L., Cassandra A.R.* Planning and Acting in Partially Observable Stochastic Domains // Artificial Intelligence. 1998. Vol. 101, No. 1–2. P. 99–134.

22. *Sicart M.* Defining game mechanics // Game studies. 2008. Vol. 8, No. 2. P. 1–14.

23. *Pnueli A.* The Temporal Logic of Programs // 18th Annual Symposium on Foundations of Computer Science. 1977. P. 46–57.

VISUAL MODELING OF GAME MECHANICS IN TULA: TEMPORAL AND PROBABILISTIC ASPECTS

V. V. Kugurakova[1] [0000-0002-1552-4910], V. T. Trofimchuk[2] [0009-0001-9106-9614]

[1, 2]*Kazan (Volga region) Federal University, Kazan, Russia*

[1]vlada.kugurakova@gmail.com, [2]vselord.beta@gmail.com

Abstract

The paper addresses the problem of visual representation of temporal and probabilistic elements in formal modeling tools for game mechanics. An analysis of existing visual modeling systems – Machinations, Petri nets, UML state diagrams, and UPPAAL – reveals their key limitations in describing complex game scenarios. An approach to visualizing temporal operators and probabilistic constructs in the Tula tool developed by the authors is proposed. A set of visual primitives is described that ensures bidirectional correspondence between visual representation and formal specification in the gameplay specification language. Examples of visual modeling of typical game scenarios with time delays, conditional probabilities, and multiplayer interac-

tions are given. The possibility of automatic transformation of visual models into formal specifications suitable for verification with PRISM and UPPAAL is demonstrated. A comparative analysis of the expressive capabilities of Tula and predecessor systems is conducted, showing that Tula is the only system among those considered that simultaneously provides formally verifiable semantics, built-in support for temporal operators (X, G, F, U), and arbitrary probability distributions, as well as a visual notation organized around resource flows and game events. The two-level specification principle with bidirectional transformation proposed in Tula closes the design–formalization–verification–correction loop, lowering the entry barrier to formal methods for practitioners without loss of verification completeness. This makes it possible to consider Tula as an intermediate layer between iterative game design and industrial verifiers.

***Keywords**: visual modeling, game mechanics, temporal operators, probabilistic models, formal verification, Tula, Machinations, Petri nets.*

Acknowledgments

This work/publication was funded by a grant from the Academy of Sciences of the Republic of Tatarstan provided to higher education institutions, scientific and other organizations to support human resource development plans in terms of encouraging their research and academic staff to defend doctoral dissertations and conduct research activities (Agreement No. 12/2025-PD-KFU dated December 22, 2025).

REFERENCES

1. *Adams E., Dormans J.* Game Mechanics: Advanced Game Design. Berkeley: New Riders, 2012. 360 p.

2. *Dormans J.* Engineering Emergence: Applied Theory for Game Design: dis. … dr. sci. Amsterdam: University of Amsterdam, 2012. 328 p.

3. *Kugurakova V.V.* Integrating temporal and probabilistic elements into the language of gameplay characteristics // Russian Internet Journal of Industrial Engineering, 2026 (accepted for publication).

4. *Rakhmankulova V.R., Kugurakova V.V.* Tula Online Tool for Balancing Video Games // Russian Digital Libraries Journal. 2025. Vol. 28, No. 4. P. 903–930.

5. *Dormans J.* Machinations: Elemental feedback patterns for game design // Proceedings of the 5th International North American Conference on Intelligent Games and Simulation (EUROSIS). 2009. P. 33–40.

6. *Dormans J.* Simulating mechanics to study emergence in games // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 2011. Vol. 7, No. 3. P. 2–7.

7. *Grünvogel S.M.* Formal Models and Game Design // Game Studies. 2005. Vol. 5, No. 1. URL: http://www.gamestudies.org/0501/gruenvogel/

8. *Peterson J.L.* Petri Net Theory and the Modeling of Systems. Prentice Hall, 1981. 290 p.

9. *Molly M.K.* Performance Analysis Using Stochastic Petri Nets // IEEE Trans. Comput. 1982. Vol. 31, No. 9. P. 913–917.

10. *Martens C.* Generative Structural Analysis of Petri Nets // Trans. on Petri Nets and Other Models of Concurrency XI. 2016. P. 37–56.

11. *Jensen K.* Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1. Berlin: Springer, 1997. 234 p.

12. About the Unified Modeling Language Specification Version 2.5.1 // OMG. OMG Unified Modeling Language (OMG UML). URL: https://www.omg.org/spec/UML/2.5.1

13. *Segala R.* Modeling and Verification of Randomized Distributed Real-Time Systems: dis. … dr. sci. MIT, 1995.

14. *Larsen K.G., Pettersson P., Yi W.* UPPAAL in a Nutshell // Int. J. Software Tools for Technology Transfer. 1997. Vol. 1, No. 1. P. 134–152.

15. *Kwiatkowska M., Norman G., Parker D.* PRISM 4.0: Verification of Probabilistic Real-Time Systems // Computer Aided Verification. 2011. P. 585–591.

16. *Alur R., Courcoubetis C., Dill D.* Model-Checking in Dense Real-Time // Information and Computation. 1993. Vol. 104, No. 1. P. 2–34.

17. *Klint P., van Rozen R.* Micro-Machinations: A DSL for Game Economies // Software Language Engineering: 6th Int. Conf., SLE 2013, Indianapolis. 2013. Vol. 8225. P. 36–55.

18. *Kugurakova V.V.* A formal approach to spatio-temporal modeling of game systems // Uchenye Zapiski Kazanskogo Universiteta. Seriya Fiziko-Matematicheskie Nauki. 2024. Vol. 166, No. 4. P. 532–554.

19. *Sahibgareeva G.F., Kugurakova V.V., Bolshakov E.S.* Game Balance Tools // Russian Digital Libraries Journal. 2023. Vol. 26, No. 2. P. 225–251.

20. *Hansson H., Jonsson B.* A Logic for Reasoning About Time and Reliability // Formal Aspects of Computing. 1994. Vol. 6, No. 5. P. 512–535.

21. *Kaelbling L.P., Littman M.L., Cassandra A.R.* Planning and Acting in Partially Observable Stochastic Domains // Artificial Intelligence. 1998. Vol. 101, No. 1–2. P. 99–134.

22. *Sicart M.* Defining game mechanics // Game studies. 2008. Vol. 8, No. 2. P. 1–14.

23. *Pnueli A.* The Temporal Logic of Programs // 18th Annual Symposium on Foundations of Computer Science. 1977. P. 46–57.

СВЕДЕНИЯ ОБ АВТОРАХ

КУГУРАКОВА Влада Владимировна – кандидат технических наук, доцент, зав. кафедрой индустрии разработки видеоигр Института информационных технологий и интеллектуальных систем Казанского федерального университета. Область научных интересов – формальные методы верификации видеоигр.

V*lada Vladimirovna KUGURAKOVA* – PhD, Head of the Department of Video Game Development Industry (GD Chair) at the Institute of Information Technology and Intelligent Systems, Kazan Federal University. Research interests include the formal methods of video game verification.

email: vlada.kugurakova@gmail.com

ORCID: 0000-0002-1552-4910

ТРОФИМЧУК Всеволод Тарасович – лаборант-исследователь кафедры индустрии разработки видеоигр Института информационных технологий и интеллектуальных систем Казанского федерального университета. Область научных интересов – использование LLM для видеоигр в реальном времени.

Vsevolod Tarasovich TROFIMCHUK – laboratory research assistant at the Department of Video Game Development Industry at the Institute of Information Technology and Intelligent Systems, Kazan Federal University. Research interests include the application of LLM in video games.

email: vselord.beta@gmail.com

ORCID:0009-0001-9106-9614

Материал поступил в редакцию 25 апреля 2026 года