DOI: 10.24892/RIJIE/20260306

Интеграция темпоральных и вероятностных элементов в язык спецификации игрового процесса*

Кугуракова В.В.
Казанский (Приволжский) федеральный университет
г. Казань, Российская Федерация
vlada.kugurakova@gmail.com

Аннотация. **В данной статье представлен новый подход к разработке языка спецификации игрового процесса, интегрирующего темпоральные и вероятностные элементы. Этот подход позволяет более точно описывать и анализировать сложные игровые механики, учитывая их временную динамику и элементы случайности. В работе рассматриваются существующие методы спецификации игровых механик, предлагается синтаксис и семантика нового языка, включающего темпоральные операторы и вероятностные конструкции. Описывается применение языка для моделирования игровых механик, анализа баланса и верификации игровых сценариев. Обсуждаются вопросы интеграции предложенного подхода с существующими инструментами разработки игр, а также его преимущества и ограничения.**

Ключевые слова: **язык спецификации, игровые механики, темпоральная логика, вероятностные модели, формальная верификация, анализ игрового баланса, разработка видеоигр.**

Введение

В современной индустрии разработки видеоигр одной из ключевых проблем является создание сбалансированного и увлекательного игрового процесса [8, 17]. Для решения этой задачи разработчики все чаще обращаются к формальным методам спецификации и верификации игровых механик [1, 7]. Однако существующие языки спецификации зачастую не учитывают в полной мере темпоральные (временные) и вероятностные аспекты игрового процесса, которые играют решающую роль в современных играх [3, 9].

Данная статья представляет новый подход к разработке языка спецификации игрового процесса, интегрирующего темпоральные и вероятностные элементы. Этот подход позволяет более точно описывать и анализировать сложные игровые механики, учитывая их временную динамику и элементы случайности.

Одним из ключевых этапов в разработке формальных методов для анализа и спецификации игровых механик является создание структурированного корпуса игровых текстов. В нашей предыдущей работе [30] мы предложили подход к созданию универсальной структуры для корпуса текстов видеоигр, который может служить основой для более сложных задач, таких как интеграция темпоральных и вероятностных элементов в спецификации игрового процесса. Данное исследование развивает эти идеи, предлагая более глубокий и специализированный подход к формальному описанию игровых механик.

Связанные работы

Перед тем как перейти к обзору существующих подходов, рассмотрим ряд связанных работ, которые внесли значительный вклад в область формализации и спецификации игровых механик.

В [25] представили подход к формальному описанию и анализу игровых механик с использованием модульных вычислительных критиков. Их работа демонстрирует возможность автоматизированного анализа игрового дизайна, что является важным шагом к формальной верификации игровых механик. Однако их подход не включает темпоральные аспекты, которые критически важны для описания динамики игрового процесса.

В [27] представлен формальный подход к описанию и комбинированию игровых механик. Их работа предлагает интересную идею рекомбинации механик, которую мы планируем адаптировать в нашем подходе. Тем не менее, их метод не учитывает вероятностные элементы, которые играют важную роль в современных играх.

В [28] представлен формальный подход к эволюционному дизайну игр, включающий спецификацию игровых правил. Их работа демонстрирует потенциал автоматизированного создания и оценки игровых механик, что мы планируем развить в нашем исследовании. Однако их подход ограничен относительно простыми настольными играми и не учитывает сложность современных видеоигр.

В [29] предложено использование генеративных грамматик для спецификации игровых механик. Эта идея интересна с точки зрения формального описания структуры игровых правил, и мы рассмотрим возможность интеграции похожего подхода в наш язык спецификации. Однако их метод не предоставляет средств для верификации созданных механик.

Работа [1] представляет собой всесторонний обзор формальных методов в разработке программного обеспечения, включая подходы, применимые к сложным интерактивным системам, таким как игры. Эта работа предоставляет ценный контекст для понимания принципов формальной верификации и спецификации, а также указывает на области, где применение этих методов может быть расширено. Мы стремимся адаптировать и развить эти идеи специфично для игровой индустрии, заполняя пробелы в существующих исследованиях по применению формальных методов в разработке игр.

* Работа выполнена за счет гранта, предоставленного Академией наук Республики Татарстан образователь-ным 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). Статья публикуется по рекомендации программного комитета Всероссийской научно-практической конференции Индустрия 4.0", https://smartindustrycon.ru

Хотя и не специфично для игровой индустрии, работа [26] представляет интересный подход к декларативной спецификации интеллектуальных агентов. Их язык спецификации, хотя и разработан для более общих целей, демонстрирует принципы, которые могут быть адаптированы для описания поведения игровых персонажей и систем искусственного интеллекта в играх. Подход, представленный в [26], к формализации логики агентов предоставляет ценные идеи для разработки языков спецификации, способных описывать сложное, контекстно-зависимое поведение, что особенно релевантных для современных игровых систем.

Наше исследование пытается объединить сильные стороны этих подходов, такие как модульность, возможность рекомбинации механик и формальное описание правил, одновременно адресуя их ограничения путем интеграции темпоральных и вероятностных элементов, а также обеспечивая возможность верификации. Кроме того, мы ставим целью создать язык спецификации, который был бы применим к широкому спектру современных видеоигр, от простых до сложных многопользовательских систем.

Обзор существующих подходов

Традиционные методы спецификации игровых механик

Для описания игровых механик исторически использовались неформальные методы, такие как текстовые описания в игровых дизайн-документах (GDD – Game Design Document – формализованное описание игрового проекта) или простые блок-схемы [12, 23]. Однако эти методы часто приводили к неоднозначности интерпретации и затрудняли автоматизированный анализ игровых систем [2, 18].

Формальные языки спецификации

В контексте формальных подходов к геймдизайну (Game Design – «игровой дизайн», процесс проектирования игровых проектов) особое место занимает фреймворк MDA (Mechanics, Dynamics, Aesthetics) [13]. Этот подход предоставляет формальную структуру для анализа и проектирования игр, разделяя их на три взаимосвязанных уровня: механики, динамику и эстетику. Хотя MDA не является языком спецификации в строгом смысле, он заложил основы для более формального подхода к описанию игровых систем и оказал значительное влияние на развитие формальных методов в геймдизайне. В последние годы наблюдается тенденция к развитию этих идей и созданию более строгих языков спецификации в игровой индустрии [3, 11]. Такие языки, как UML (Unified Modeling Language) и его расширения для игр, позволяют более структурированно описывать игровые системы [7, 24], продолжая линию формализации, начатую MDA.

Темпоральная логика в спецификации систем

Темпоральная логика, широко используемая в верификации программных систем, предоставляет мощный инструментарий для описания временных свойств систем [6, 15]. Однако её применение в контексте игровых механик остается ограниченным, во многом из-за сложности интеграции с другими аспектами игрового дизайна [10].

Вероятностные модели в игровом дизайне

Вероятностные модели в игровом дизайне часто опираются на фундаментальные принципы теории игр, которая предоставляет математический аппарат для анализа стратегических взаимодействий. В фундаментальном труде по теории игр [5] отмечается, что вероятностные стратегии и исходы являются ключевыми концепциями при моделировании ситуаций с неполной информацией или случайными событиями. Эти принципы находят широкое применение в игровом дизайне, особенно при разработке систем искусственного интеллекта противников, моделировании экономических систем игры и балансировке игровых механик. Вероятностные модели, такие как марковские цепи и стохастические сети Петри, успешно применяются для моделирования некоторых аспектов игрового процесса, особенно в области балансировки [4, 16, 22].

Предлагаемый подход

Предлагаемый язык спецификации основан на следующих ключевых принципах:

1) Интеграция темпоральных операторов для описания временных зависимостей в игровых механиках;
2) Введение вероятностных конструкций для моделирования случайных событий и неопределенностей;
3) Визуальное представление спецификаций для облегчения понимания и анализа;
4) Модульность и расширяемость для адаптации к различным жанрам игр;
5) Использование универсальной структуры корпуса текстов видеоигр [30] для эффективного извлечения и анализа игровых элементов.

Синтаксис и семантика языка

При разработке синтаксиса и семантики нашего языка спецификации мы опирались на фундаментальные работы в области определения игровых механик. В частности, в [14] было предложено рассматривать игровые механики как *'методы, вызываемые агентами для взаимодействия с игровым миром'*. Это определение подчеркивает важность чёткого описания действий и взаимодействий в игре. Развивая эту идею, мы разработали язык, который позволяет формально описывать не только сами механики, но и их темпоральные и вероятностные аспекты. Наш подход расширяет концепцию [14], предоставляя более богатый инструментарий для спецификации сложных игровых систем.

Для обеспечения ясности и однозначности в описании игровых механик будем использовать следующие базовые конструкции:

1) Сущности (Entity) – представляют игровые объекты, персонажей, ресурсы;
2) Состояния (State) – описывают возможные состояния сущностей;
3) Событие (Event) – представляют действия или изменения в игровом мире, состоит из следующих элементов:
- Requires – предусловие для выполнения события,
- Effect – результат выполнения события,
- P[...] = x – вероятность наступления определенного эффекта, где $x \in [0, 1]$;
4) Правила (Rule) – определяют логику переходов между состояниями и реакции на события, включает:
- When – условие применения правила,
- Effect – результат применения правила;
5) Цель (Target) – объект, к которому применяется действие.

Темпоральные операторы:

1) X (Next) – описывает, что должно произойти в следующий момент времени;

2) F (Future) – указывает, что событие должно произойти в какой-то момент в будущем;
3) F(φ): φ должно стать истинным в некотором будущем состоянии;
4) G (Globally) – определяет свойство, которое должно выполняться всегда;
5) G(φ): Глобально. φ должно быть истинным во всех состояниях.
6) U (Until) – описывает условие, которое должно выполняться до наступления определенного события. φ U ψ: φ должно быть истинным до тех пор, пока не станет истинным ψ.

Пример использования:

G(PlayerHealth > 0 U GameOver)

– это выражение означает, что здоровье игрока должно оставаться положительным до наступления события окончания игры.

Для описания вероятностных аспектов вводятся следующие конструкции:
1) P[expr] – вероятность выполнения выражения expr;
2) Rand(min, max) – генерация случайного числа в заданном диапазоне;
3) Dist(type, params) – определение вероятностного распределения.

Пример использования:

P[F(PlayerWins)] >= 0.6

– это выражение означает, что вероятность того, что игрок выиграет в будущем, должна быть не менее 0.6.

Язык позволяет комбинировать темпоральные и вероятностные элементы, что дает возможность описывать сложные игровые сценарии:

P[G(PlayerHealth > 50)] >= 0.7

– это выражение означает, что вероятность того, что здоровье игрока всегда будет выше 50, должна быть не менее 0.7.

Вводятся также логические операторы (∧ – логическое И, ∨ – логическое ИЛИ, -> – импликация (если... то) и специальные функции:
1) Min(a, b) – возвращает минимальное из двух значений%
2) Enum(...) – определяет перечисление возможных значений;
3) List<T> – определяет список элементов типа T.

Пример использования:

```
Entity Card {
  State {
    Element: Enum(Fire, Water, Earth, Air)
    Attack: Int[1, 10]
    Health: Int[1, 10]
  }
  Event Attack {
    Target: Card
    Effect: Target.Health -= Self.Attack
    P[Self.Health -= Target.Attack] = 0.5
  }
}
```

В этом примере определена сущность карта (Card) с состоянием, включающим элемент, атаку (Attack) и здоровье (Health). Событие Attack описывает атаку одной карты другой, где целевая карта всегда получает урон, а атакующая карта имеет 50% шанс получить ответный урон. Использование этих конструкций позволяет создавать четкие и однозначные спецификации игровых механик, которые могут быть легко переведены в исполняемый код или использованы для формального анализа и верификации игровой логики.

Формальное определение каждого оператора

Пусть Φ – множество всех формул нашего языка, S – множество всех возможных состояний системы, и $\pi : N \to S$ – путь выполнения системы.

Темпоральные операторы:
X (Next): $(\pi, i) \models X\,\varphi \Leftrightarrow (\pi, i+1) \models \varphi$,
F (Future): $(\pi, i) \models F\,\varphi \Leftrightarrow \exists j \geq i : (\pi, j) \models \varphi$,
G (Globally): $(\pi, i) \models G\,\varphi \Leftrightarrow \forall j \geq i : (\pi, j) \models \varphi$,
U (Until): $(\pi, i) \models \varphi\, U\, \psi \Leftrightarrow \exists j \geq i : ((\pi, j) \models \psi \wedge \forall k, i \leq k < j : (\pi, k) \models \varphi)$.

Вероятностные операторы:
P[expr]: P[expr] ≥ p ⇔ Вероятность выполнения expr не менее p,
Rand(min, max): генерирует случайное число в интервале [min, max],
Dist(type, params): определяет вероятностное распределение заданного типа с параметрами params.

Точные правила вычисления для каждого оператора
1) X φ: Проверяем φ в следующем состоянии пути;
2) F φ: Итеративно проверяем φ в каждом последующем состоянии, пока не найдем удовлетворяющее или не достигнем конца пути;
3) G φ: Проверяем φ в текущем и каждом последующем состоянии. Если находим несоответствие, возвращаем «ложь»;
4) φ U ψ: Проверяем φ в каждом состоянии, пока не найдем состояние, удовлетворяющее ψ.

Граничные случаи

Для X φ: если текущее состояние последнее, X φ всегда ложно.

Для F φ и φ U ψ: если путь конечен и условие не выполняется до конца пути, формула ложна.

Для G φ: если путь бесконечен, мы можем использовать конечное представление с циклом для проверки.

Формальное описание взаимодействия между различными операторами

Дистрибутивность:

$$X(\varphi \wedge \psi) \equiv X\,\varphi \wedge X\,\psi,$$
$$F(\varphi \vee \psi) \equiv F\,\varphi \vee F\,\psi,$$
$$G(\varphi \wedge \psi) \equiv G\,\varphi \wedge G\,\psi.$$

Двойственность:

$$\neg X\,\varphi \equiv X\,\neg\varphi,$$
$$\neg F\,\varphi \equiv G\,\neg\varphi,$$
$$\neg G\,\varphi \equiv F\,\neg\varphi.$$

Вложенность:

$$F\,F\,\varphi \equiv F\,\varphi,$$
$$G\,G\,\varphi \equiv G\,\varphi,$$
$$F\,G\,F\,\varphi \equiv G\,F\,\varphi.$$

Взаимодействие с вероятностными операторами:

$$P[F\,\varphi] + P[G\,\neg\varphi] = 1,$$
$$P[\varphi\, U\, \psi] = P[F\,(\psi \wedge (\neg\varphi\, U\, \psi))].$$

Математические модели для вероятностных и темпоральных аспектов

1) Темпоральная логика:

Используем линейную темпоральную логику (LTL) для описания свойств путей выполнения

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid X\,\varphi \mid \varphi\, U\, \psi,$$

где p – атомарное высказывание.

2) Вероятностная модель:

Используем дискретные марковские цепи (DTMC) для моделирования вероятностного поведения

$$DTMC = (S, P, L),$$

где S – конечное множество состояний, $P : S \times S \to [0, 1]$ – функция вероятности перехода, $L : S \to 2^{AP}$ – функция пометки состояний атомарными высказываниями.

3) Комбинированная модель:

Для объединения темпоральных и вероятностных аспектов используем вероятностную темпоральную логику (PCTL)

$\varphi ::= true \mid a \mid \varphi \wedge \varphi \mid \neg\varphi \mid P \bowtie p[\psi]$ $\psi ::= X\,\varphi \mid \varphi\, U_{\leq k}\, \varphi \mid \varphi\, U\, \varphi$,

где a – атомарное высказывание, $\bowtie \in \{<, \leq, \geq, >\}$, $p \in [0, 1]$, $k \in N$.

Эта формальная семантика обеспечивает точное и однозначное определение операторов языка, что критически важно для корректной интерпретации спецификаций и реализации инструментов анализа и верификации.

Базовые конструкции языка спецификации

Для обеспечения ясности и однозначности в описании игровых механик, наш язык спецификации использует *ряд ключевых конструкций*. Ниже приведено описание основных элементов языка и их семантика:

1) Entity: описывает игровой объект или участника. Например, Card или Player;

2) State: определяет набор атрибутов, характеризующих состояние сущности. Пример: Health: Int[0, 30] означает здоровье в диапазоне от 0 до 30;

3) Event: описывает действие или происшествие в игре. Состоит из следующих элементов:

- Requires: предусловие для выполнения события,
- Effect: результат выполнения события,
- P[...] = x: вероятность наступления определенного эффекта, где $x \in [0, 1]$;

4) Rule: определяет правило игры, применяемое при определенных условиях. Включает:

- When: условие применения правила,
- Effect: результат применения правила;

5) Target: указывает на объект, к которому применяется действие;

6) Темпоральные операторы:

- G(φ): глобально. φ должно быть истинным во всех состояниях,
- F(φ): в будущем. φ должно стать истинным в некотором будущем состоянии,
- φ U ψ: до тех пор, пока. φ должно быть истинным до тех пор, пока не станет истинным ψ;

7) Логические операторы:

- ∧: логическое И,
- ∨: логическое ИЛИ,
- ->: Импликация (если... то);

7) Специальные функции:

- Min(a, b): возвращает минимальное из двух значений,
- Enum(...): определяет перечисление возможных значений,
- List<T>: определяет список элементов типа T.

Пример использования:

```
Entity Card {
  State {
    Element: Enum(Fire, Water, Earth, Air)
    Attack: Int[1, 10]
    Health: Int[1, 10]
  }

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

В этом примере мы определяем сущность Card с состоянием, включающим элемент, атаку и здоровье. Событие Attack описывает атаку одной карты другой, где целевая карта всегда получает урон, а атакующая карта имеет 50% шанс получить ответный урон.

Использование этих конструкций позволяет создавать четкие и однозначные спецификации игровых механик, которые могут быть легко переведены в исполняемый код или использованы для формального анализа и верификации игровой логики.

Применение языка спецификации

Моделирование игровых механик

Использование структурированного корпуса текстов видеоигр [30] значительно облегчает процесс моделирования игровых механик и анализа баланса. Например, при моделировании боевой системы, описанной ниже, мы можем обратиться к корпусу для извлечения типичных параметров здоровья и силы персонажей в играх аналогичного жанра. Это позволяет установить начальные значения для нашей модели, основываясь на проверенных данных из существующих игр. Более того, анализ поля 'comment' в корпусе может предоставить информацию о типичных вероятностях успешных атак или критических ударов, что помогает в настройке вероятностных элементов нашей спецификации.

Для иллюстрации применения предложенного языка спецификации, рассмотрим конкретный пример. Ключевые конструкции (Entity, State, Event, Rule) позволяют структурированно описывать различные аспекты игровой механики, сочетая в себе как статические характеристики игровых объектов, так и динамические аспекты их взаимодействия.

Листинг 1

Пример моделирования механики боевой системы в ролевой игре

```
1   Entity Player {
2     State {
3       Health: Int [0, 100]
4       Strength: Int [1, 20]
5     }
6
7     Event Attack {
8       Damage = Rand(1, Strength)
9       P[Enemy.Health -= Damage] = 0.8
10      P[Self.Health -= Rand(1, 5)] = 0.2
11    }
12  }
13
14  Entity Enemy {
15    State {
```

```
16        Health: Int [0, 50]
17        Power: Int [1, 10]
18     }
19
20     Event CounterAttack {
21        Triggered: Player.Attack
22        P[Player.Health -= Rand(1, Power)] = 0.6
23     }
24  }
25
26  Rule CombatFlow {
27     G(Player.Health > 0 ∧ Enemy.Health > 0 U GameOver)
28     F(Enemy.Health <= 0) -> F(LevelComplete)
29     P[F(Player.Health <= 0)] < 0.3
30  }
31
```

Представленный пример (см. лист. 1) демонстрирует, как язык позволяет описывать состояния игровых сущностей, события с вероятностными исходами и правила, определяющие общий ход игрового процесса с использованием темпоральных операторов.

Анализ баланса

Предложенный язык спецификации позволяет проводить автоматизированный анализ баланса игровых механик. Например, можно проверить, удовлетворяет ли спецификация следующим критериям баланса:

1) Вероятность победы игрока должна быть в определенном диапазоне

$$0.4 <= P[F(Enemy.Health <= 0)] <= 0.7;$$

2) Средняя продолжительность боя должна быть в определенных пределах

$$P[F[10, 30](Enemy.Health <= 0 \vee Player.Health <= 0)] >=0.8;$$

3) Вероятность критических ситуаций не должна быть слишком высокой

$$P[G(Player.Health > 20)] >= 0.6.$$

Таким образом, предложенный язык спецификации предоставляет мощный инструментарий для формального анализа баланса игровых механик. Возможность точно формулировать и автоматически проверять критерии баланса позволяет разработчикам быстро итерировать дизайн, выявлять потенциальные проблемы на ранних стадиях разработки и обеспечивать более сбалансированный и увлекательный игровой опыт. Этот подход особенно ценен для сложных игровых систем, где традиционные методы балансировки могут быть недостаточно эффективными или слишком трудоемкими.

Предложенный подход к анализу баланса опирается на наши предыдущие исследования в этой области. В частности, в [21] были предложены методы генерации и балансирования игровых механик видеоигр, которые могут быть расширены и формализованы с использованием нашего языка спецификации. Интеграция их подхода с нашим формальным языком позволяет более точно определять критерии баланса и автоматизировать процесс их проверки.

Верификация игровых сценариев

Язык спецификации также позволяет проводить формальную верификацию игровых сценариев. Например, можно проверить такие свойства, как достижимость целей F(LevelComplete), отсутствие тупиковых ситуаций G(F(PlayerCanAct)) или справедливость G(F(PlayerAdvantage) ∧ F(EnemyAdvantage)) – в рамках игрового процесса.

Формальная верификация игровых сценариев с использованием предложенного языка спецификации открывает новые возможности для обеспечения качества и целостности игрового опыта. Возможность автоматически проверять ключевые свойства игры, такие как достижимость целей, отсутствие тупиковых ситуаций и справедливость игрового процесса, позволяет разработчикам создавать более надежные и удовлетворительные игровые системы. Это не только повышает качество конечного продукта, но и потенциально сокращает время и ресурсы, необходимые для тестирования и отладки игровых механик. В результате, формальная верификация становится неотъемлемой частью процесса разработки, дополняя традиционные методы тестирования и обеспечивая более глубокое понимание поведения игровой системы.

Демонстрация полного цикла разработки

Для иллюстрации применения предложенного подхода рассмотрим полный цикл разработки небольшой, но нетривиальной игры. Эта игра должна сочетать в себе элементы стратегии, случайности и балансировки, чтобы позволить продемонстрировать различные аспекты нашего языка спецификации и методологии разработки. Итак, «Элементальные Дуэли» – это карточная игра для двух игроков, где каждый игрок управляет колодой элементальных существ (огонь, вода, земля, воздух). Цель игры – уменьшить здоровье противника до нуля.

Спецификация игровых механик

Для спецификации игровых механик будем использовать разработанный нами инструмент Tula [16], который представляет собой онлайн-платформу для балансировки игрового процесса. Tula основан на принципах, схожих с Machinations.io [12], но предлагает ряд улучшений, включая более интуитивный интерфейс и расширенные аналитические возможности.

Используя наш язык спецификации, опишем основные элементы и правила игры (см. лист. 2).

Листинг 2

Элементы и правила игры

```
1   Entity Card {
2     State {
3       Element: Enum(Fire, Water, Earth, Air)
4       Attack: Int[1, 10]
5       Health: Int[1, 10]
6     }
7     Event Play {
8       Requires: Player.Mana >= 1
9       Effect: Player.Mana -= 1
10      P[Board.AddCard(Self)] = 1
11    }
12    Event Attack {
13      Target: Card
14      Effect: Target.Health -= Self.Attack
15      P[Self.Health -= Target.Attack] = 0.5
16    }
17  }
18  Entity Player {
19    State {
20      Health: Int[0, 30]
21      Mana: Int[0, 10]
22      Deck: List<Card>
23      Hand: List<Card>
24    }
```

```
  Event DrawCard {
    Requires: Deck.Size > 0
    Effect: Hand.Add(Deck.RemoveTop())
  }
  Event StartTurn {
    Effect: Mana = Min(Mana + 1, 10)
    Trigger: DrawCard
  }
}
Rule ElementalAdvantage {
  When: Card(Element: Fire).Attack(Card(Element: Air))
  Effect: Attacker.Attack *= 1.5
}
Rule GameFlow {
  G(Player1.Health > 0 ∧ Player2.Health > 0 U GameOver)
  F(Player1.Health <= 0 ∨ Player2.Health <= 0) -> Game-
Over
}
```

Используя Tula, создадим диаграмму, отражающую основные элементы и правила игры. На рис. 1 представлен граф игры. Некоторые элементы – «правила» (красные прямоугольники) – могут влиять как на конкретную сущность (Card), так и на игру в целом. Есть «сущности» (зеленые прямоугольники) – это объекты, которые участвуют в процессе игры. И «события» (фиолетовые прямоугольники) – элементы, которые ведут к изменению состояний.

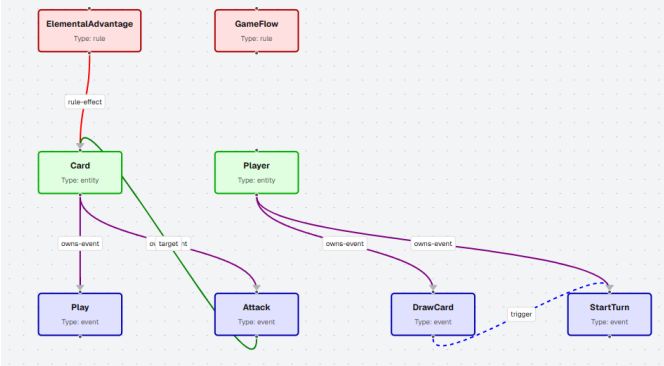


Рис. 1. Верхнеуровневая визуализация, отображающая отдельные элементы и их связи в соответствии с языком спецификаций игры

Визуализация структуры

Tula предоставляет возможность автоматической визуализации структуры игры в виде графа, что облегчает понимание взаимосвязей между различными элементами игры. Для удобства понимания и анализа данных существует разделение на уровни детализации. На верхнем уровне находится общий граф игры (см. рис. 1), на нижнем уровне находится граф атомарных операций, который производит симуляции и анализ данных. Концепция: Entity (верхний уровень) – это «чёрный ящик», чьи внутренние переменные и состояния (States) реализованы как сеть узлов Pool (нижний уровень визуализации). На рис. 2 представлен отрывок нижнеуровневого графа, ответственный за вычисление поведения колоды карт.

Верификация игровых сценариев

Для верификации игровых сценариев используются инструменты PRISM [15] и UPPAAL [31]. Tula позволяет экспортировать модель игры в форматы, совместимые с этими инструментами.

В PRISM можно проверить вероятностные свойства:

- вероятность победы первого игрока

P=? [F Player1.Health <= 0],

- вероятность затяжной игры

P=? [G[0,20] (Player1.Health > 0 & Player2.Health > 0)].

В UPPAAL анализируются темпоральные свойства:

- проверка, что игра всегда заканчивается

A<> GameOver,

- проверка чередования ходов

A[] (Player1Turn -> A<> Player2Turn).

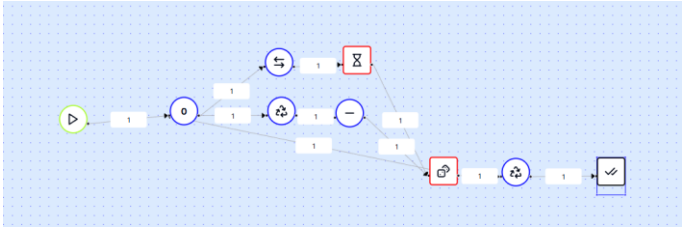
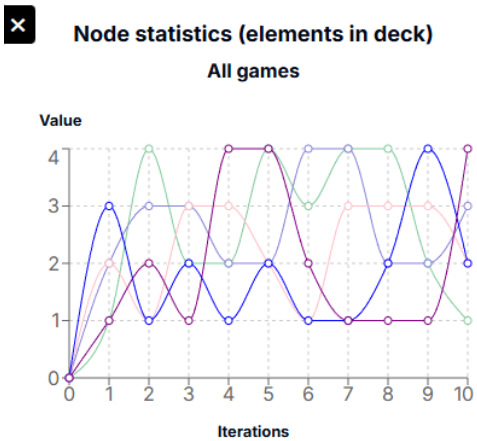


Рис. 2. Эмуляция поведения колоды карт

Анализ баланса

Tula предоставляет встроенные инструменты для анализа баланса (см., например, рис. 3). Мы проводим серию симуляций с различными параметрами, анализируя следующие аспекты: распределение элементов в колодах, соотношение атаки и здоровья карт, влияние элементальных преимуществ на исход игры.



Metrics	Value
Average value (AVR)	2.4
Median (MEDIAN)	2.5
Minimum value (MIN)	1
Maximum value (MAX)	4
Standard Deviation (STD)	3.5
Range (RANGE)	1

Рис. 3. Игровая статистика

Заключение

Интеграция темпоральных и вероятностных элементов в язык спецификации игрового процесса представляет собой значительный шаг вперед в области формальных методов разработки игр. Предложенный подход открывает новые возможности для более точного и комплексного описания сложных игровых механик, учитывая их временную динамику и элементы случайности.

Четкое определение формальной семантики операторов языка существенно усиливает теоретическую базу предложенного подхода. Это улучшает возможности для автоматической верификации и анализа игровых механик, позволяя более точно предсказывать поведение игровых систем на ранних этапах разработки. Однако стоит отметить, что сложность формальной семантики может представлять трудности для разработчиков игр без глубокого математического образования. Для решения этой проблемы мы планируем разработать интуитивно понятные визуальные представления и упрощенные объяснения ключевых концепций.

Связь между формальной семантикой и практической реализацией игровой логики в прототипе является одним из ключевых аспектов нашего подхода. Четкое математическое определение операторов должно облегчать процесс трансляции спецификаций в исполняемый код, тогда это открывает возможности для автоматизации процесса разработки и уменьшения количества ошибок при реализации игровых механик.

В области анализа и верификации открываются несколько перспективных направлений исследований. Ключевой задачей является разработка эффективных алгоритмов проверки моделей, способных работать со спецификациями, включающими темпоральные и вероятностные элементы. Это позволит более точно анализировать сложные игровые системы. Параллельно с этим, важным направлением является создание методов автоматической генерации тестовых сценариев на основе спецификаций, что может значительно ускорить и улучшить процесс тестирования игр. Кроме того, многообещающим выглядит исследование возможностей применения методов машинного обучения для анализа и оптимизации игровых механик, описанных на предложенном языке. Такой подход может открыть новые возможности для автоматизированной балансировки и улучшения игрового дизайна, основанного на формальных спецификациях.

Литература

1. Almeida J.B. Rigorous software development: an introduction to program verification / J.B. Almeida, M.J. Frade, J.S. Pinto, S.M. de Sousa. – Springer Science & Business Media, 2011. – 298 p.

2. Dormans J. Engineering emergence: applied theory for game design: dis. … dr. sci.– University of Amsterdam, 2012. – 328 p.

3. Grünvogel S.M. Formal models and game design / S.M. Grünvogel // Game Studies. – 2005. – Vol. 5, № 1. – URL: http://www.gamestudies.org/0501/gruenvogel/

4. Jaffe A. Evaluating competitive game balance with restricted play / A. Jaffe, A. Miller, E. Andersen, Y.E. Liu, A. Karlin, Z. Popovic // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2012. – Vol. 8, № 1. – P. 26-31.

5. Osborne M.J. A course in game theory / M.J. Osborne, A. Rubinstein. – MIT Press, 1994. – 352 p.

6. Pnueli A. The temporal logic of programs // 18th IEEE Annual Symposium on Foundations of Computer Science. – 1977. – P. 46-57.

7. Smith A.M. Ludocore: a logical game engine for modeling videogames / A.M. Smith, M.J. Nelson, M. Mateas // Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games. – 2010. – P. 91-98.

8. Salen K. Rules of play: game design fundamentals / K. Salen, K.S. Tekinbaş, E. Zimmerman. – MIT Press, 2004. – 688 p.

9. Togelius J. Search-based procedural content generation: a taxonomy and survey / J. Togelius, G.N. Yannakakis, K.O. Stanley, C. Browne // IEEE Transactions on Computational Intelligence and AI in Games. – 2011. – Vol. 3, № 3. – P. 172-186.

10. Verbrugge C. A structure for modern computer narratives // Computational Logic in Multi-Agent Systems. – Berlin, Heidelberg: Springer, 2009. – P. 308-322.

11. Zook A. Automatic game design via mechanic generation / A. Zook, M.O. Riedl // Twenty-Eighth AAAI Conference on Artificial Intelligence. – 2014. – P. 530-536.

12. Adams E. Game mechanics: advanced game design / E. Adams, J. Dormans. – New Riders, 2012. – 496 p.

13. Hunicke R. MDA: a formal approach to game design and game research / R. Hunicke, M. LeBlanc, R. Zubek // Proceedings of the AAAI Workshop on Challenges in Game AI. – 2004. – Vol. 4, № 1. – P. 1-5.

14. Sicart M. Defining game mechanics // Game Studies. – 2008. – Vol. 8, № 2. – P. 1-14.

15. Kwiatkowska M. PRISM 4.0: verification of probabilistic real-time systems / M. Kwiatkowska, G. Norman, D. Parker // Computer Aided Verification. – Berlin, Heidelberg: Springer, 2011. – P. 585-591.

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

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

18. Сахибгареева Г.Ф. Редактор интерактивной структуры для инструмента генерации сценарных прототипов / Г.Ф. Сахибгареева, В.В. Кугуракова // Электронные библиотеки. – 2022. – Т. 24, № 6. – С. 1184-1202.

19. Шубин А.В. Обзор практик управления проектами в игровой разработке / А.В. Шубин, Г.Ф. Сахибгареева, В.В. Кугуракова // Электронные библиотеки. – 2022. – Т. 25, № 5. – С. 533-552.

20. Шараева Р.А. Методика упрощения таск-трекинга в проектах игровой индустрии / Р.А. Шараева, В.В. Кугуракова, Н.Э. Селезнева // Программные продукты и системы. – 2022. – Т. 35, № 3. – С. 374-383.

21. Сахибгареева Г.Ф. Генерация и балансирование игровых механик видеоигр / Г.Ф. Сахибгареева, В.В. Кугуракова, Э.С. Большаков // Научный сервис в сети Интернет: труды XXIV Всероссийской научной конференции. – М.: ИПМ им. М.В. Келдыша, 2022. – С. 455-485.

22. Сахибгареева Г.Ф. Практики балансирования компьютерных игр / Г.Ф. Сахибгареева, В.В. Кугуракова // Программные системы: теория и приложения. – 2022. – Т. 13, № 3. – С. 255-273.

23. Кугуракова В.В. Раскадровка как одно из представлений сценарного прототипа компьютерных игр / В.В. Кугуракова, Г.Ф. Сахибгареева, О.А. Бедрин // Электронные библиотеки. – 2021. – Т. 24, № 2. – С. 408-444.

24. Sahibgareeva G.F. Branched structure component for a video game scenario prototype generator / G.F. Sahibgareeva, V.V. Kugurakova // CEUR Workshop Proceedings. – 2021. – Vol. 3066. – P. 101-111.

25. Osborn J.C. Modular computational critics for games / J.C. Osborn, A. Grows, M. Mateas // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2017. – Vol. 13, № 1. – P. 1-7.

26. Bex F. A declaration language for intelligent agents / F. Bex, K. Raymond // Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning. – Cham: Springer, 2018. – P. 3-18.

27. Nelson M.J. Recombinable game mechanics for automated design support / M.J. Nelson, M. Mateas // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. – 2008. – Vol. 4, № 1. – P. 1-6.

28. Browne C. Evolutionary game design / C. Browne, F. Maire // IEEE Transactions on Computational Intelligence and AI in Games. – 2010. – Vol. 2, № 1. – P. 1-16.

29. Martens C. Generative grammars for specifying game mechanics / C. Martens, P. Mawhorter, S. Warter // Proceedings of the 15th International Conference on the Foundations of Digital Games. – 2020. – P. 1-4.

30. Нурлыгаянов Н.Р. Подход к созданию корпуса текстов видеоигр на основе универсальной структуры / Н.Р. Нурлыгаянов, В.В. Кугуракова // Электронные библиотеки. – 2024. – Т. 27, № 4. – С. 578-597.

31. Behrmann G. A tutorial on Uppaal 4.0 / G. Behrmann, A. David, K.G. Larsen. – Updated November 28, 2006. – URL: https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

DOI: 10.24892/RIJIE/20260306

Integrating Temporal and Probabilistic Elements into the Language of Gameplay Characteristics

Kugurakova V.V.
Kazan Federal University
Kazan, Russian Federation
vlada.kugurakova@gmail.com

Abstract. **This article presents a new approach to developing a gameplay feature language that integrates temporal and probabilistic elements. This approach allows for a more accurate description and analysis of complex game mechanics while preserving their temporal dynamics and randomness. The study examines existing methods for defining slot machine mechanics and proposes the syntax and semantics of a new language, including temporal operators and probabilistic constructs. The article explores the application of the language for modeling game mechanics, analyzing balance, and verifying game scenarios. The paper discusses how the proposed approach compares to existing game development tools, as well as its advantages and limitations.**

Keywords**: specification language, game mechanics, temporal logic, probabilistic models, formal verification, game balance analysis, video game development.**

Библиографическое описание статьи

Кугуракова В.В. Интеграция темпоральных и вероятностных элементов в язык спецификации игрового процесса // Машиностроение: сетевой электронный научный журнал. – 2026. – Т.13, №3. – С. 34-41. DOI: 10.24892/RIJIE/20260306

Reference to article

Kugurakova V.V. Integrating temporal and probabilistic elements into the language of gameplay characteristics, *Russian Internet Journal of Industrial Engineering*, 2026, vol.13, no.3, pp. 34-41. DOI: 10.24892/RIJIE/20260306