System, Integration, and Unit Test Plan for EMgine: A Computational Model of Emotion for Enhancing Non-Player Character Believability in Games

Geneva M. Smith

Version 0.1 (April 20, 2023)

Revision History

Date	Version	Notes
April 20, 2023	0.1	• Initial document with the Emotion Intensity Component (Emotion Intensity Type [M1] and Emotion State Type [M3] Modules) based on SRS Version 1.5.1, MG Version 1.5, and MIS Version 0.1.1

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations and Acronyms

For EMgine's other symbols, abbreviations, and acronyms, see the:

- Software Requirement Specification (SRS) at https://github.com/GenevaS/EMgine/blob/main/docs/SRS/EMgine_SRS.pdf, and
- Module Guide (MG) at https://github.com/GenevaS/EMgine/blob/main/docs/Design/MG/EMgine_MG.pdf, and
- Module Interface Specification (MIS) at https://github.com/GenevaS/EMgine/blob/main/docs/Design/MIS/EMgine_MIS.pdf.

Abbrv.	Description
ATP	Acceptance Test Plan
CME	Computational Model of Emotion
IDE	Integrated Development Environment
M	Module defined in the MG
MG	Module Guide
MIS	Module Interface Specification
MTP	Master Test Plan
NF	Nonfunctional Requirement defined in the SRS
NPC	Non-Player Character (Video Games)
R	Functional Requirement defined in the SRS
SDA	Software Development Artifact
SIUTP	System, Integration, and Unit Test Plan
SDLC	Software Development Life Cycle
SRS	Software Requirements Specification
T	Test
V & V	Verification and Validation
VS	Microsoft Visual Studio

2 Introduction

This document describes the System, Integration, and Unit Test Plan (SIUTP) for EMgine. Its purpose is to describe the testing efforts to verify the ability of EMgine's implementation to meet its functional and nonfunctional requirements as described in its Software Requirements Specification (SRS, Version 1.5.1) and additional data types and models as described in its Module Guide (MG, Version 1.5) and Module Interface Specification (MIS, Version 0.1.1). This also acts as a basis for regression testing as EMgine is developed further. It describes tests for each requirement and module, traceability between tests and EMgine's requirements, and testing and verification tools.

This test plan does *not* include validation efforts to evaluate EMgine's acceptability for its end-users and stakeholders. That information is in its Acceptance Test Plan (ATP).

The document's content and organization is loosely based on IEEE Std 829-2008 Clause 9: Level Test Plan (LTP) (IEEE Computer Society, 2008).

2.1 Summary of EMgine's Purpose and Design Goals

EMgine is a Computational Model of Emotion (CME) for Non-Player Characters (NPCs) to enhance their believability, with the goal of improving long-term player engagement. EMgine is for *emotion generation*, accepting user-defined information from a game environment to determines what emotion and intensity a NPC is "experiencing". How the emotion is expressed and what other effects it could have on game entities is left for game designers/developers to decide.

EMgine aims to provide a feasible and easy-to-use method for game designers/developers to include emotion in their NPCs, they perceive to be challenging with the current tools and restrictions (Broekens et al., 2016). EMgine should be modular and portable such that game designers/developers can use it in their regular development environment, and should not require knowledge of affective science, psychology, and/or emotion theories. Therefore, it is a library of components to maximize a game designer/developer's control over how and when EMgine functions.

2.2 Scope of Testing Efforts

The overall goals of EMgine's system, integration, and unit testing effort are:

1. Ensure traceability between EMgine's verification efforts and its SRS, MG, and MIS
2. Build confidence in the correctness and accuracy of EMgine's source code
3. Build confidence in EMgine's overall verification efforts

This test plan must be *reviewed* for each new major version of EMgine's SRS, MG, and/or MIS and revised accordingly. This ensures that these testing efforts remain relevant throughout EMgine's development. For new minor versions of EMgine's SRS, MG, and/or MIS, the review can be limited to only those sections related to SRS, MG, and/or MIS modifications to help reduce the effort required when testing the next major version.

This test plan must be *executed* in full for each new major version of EMgine's source code. Previous versions do not need to be retested. The SIUTP can be partially executed for new minor versions of the source code to help reduce the effort required when testing the next major version.

The SIUTP describes the "Dynamic Testing Method" for Implementation Verification referenced in EMgine's Master Test Plan (MTP) Section 3.5.

2.3 Location in the Software Development Life Cycle

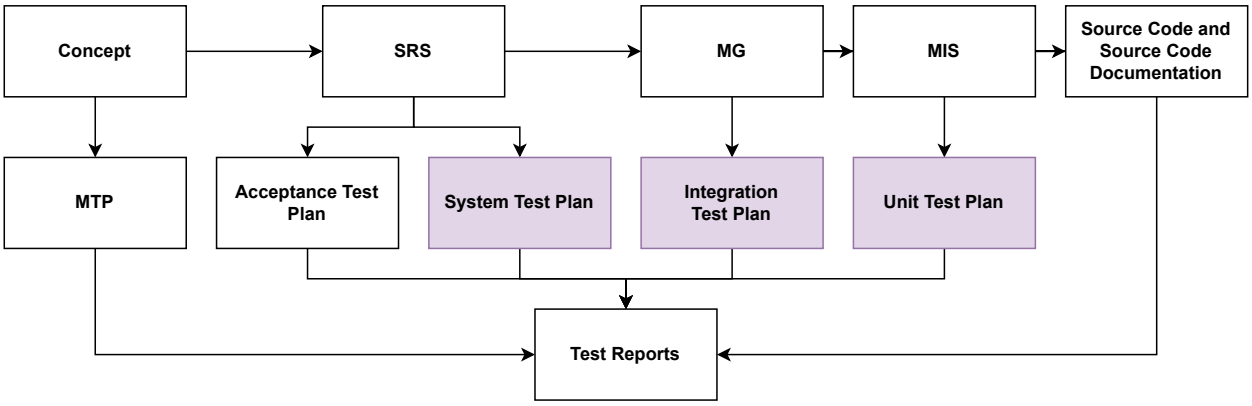

Figure 1: Dependencies Between EMgine's SDAs (SIUTP Highlighted)

2.4 Relevant Documentation

The System, Integration, and Unit Test Plan (SIUTP) refers to the following Software Development Artifacts (SDAs):

- **Title**: Master Test Plan for EMgine: A Computational Model of Emotion for Enhancing Non-Player Character Believability in Games (Version 1.0)
  Location: https://github.com/GenevaS/EMgine/blob/main/docs/TestPlans/MTP/EMgine_MTP.pdf
  Description: Description of EMgine's test planning and management, including the overall goals of the testing efforts, how they relate to EMgine's concept, and where they fit in EMgine's Software Development Life Cycle (SDLC).

- **Title**: Software Requirements Specification for EMgine: A Computational Model of Emotion for Enhancing Non-Player Character Believability in Games (Version 1.5.1)
  Location: https://github.com/GenevaS/EMgine/blob/main/docs/SRS/EMgine_SRS.pdf
  Description: EMgine's specifications as a self-contained system that interfaces with external systems including necessary functionality and the underlying assumptions and models.

- **Title**: Module Guide for EMgine: A Computational Model of Emotion for Enhancing Non-Player Character Believability in Games (Version 1.5)
  Location: https://github.com/GenevaS/EMgine/blob/main/docs/Design/MG/EMgine_MG.pdf
  Description: Description of EMgine's modular structure and architectural decisions.

- **Title**: Module Interface Specification for EMgine: A Computational Model of Emotion for Enhancing Non-Player Character Believability in Games (Version 0.1.1)
  Location: https://github.com/GenevaS/EMgine/blob/main/docs/Design/MIS/EMgine_MIS.pdf
  Description: Mathematically-based specifications of EMgine's modules.

This document also refers to the:

- IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998 (R2009)) (IEEE Computer Society, 2009) to inform the evaluation of EMgine's Software Requirements Specification (SRS)
- ISO/IEC/IEEE International Standard - Systems and software engineering – System life cycle processes (ISO/IEC/IEEE 15288:2015(E)) (ISO, IEC, and IEEE, 2015) to inform the evaluation of EMgine's design
- IEEE Standard for System, Software, and Hardware Verification and Validation (IEEE Std 1012-2016) (IEEE Computer Society, 2017) to inform the creation of this document and the System, Integration, and Unit Test Plan
- IEEE Standard for System, Software, and Hardware Verification and Validation (IEEE Std 1012-2016) (IEEE Computer Society, 2008) to inform the creation of this document, the System, Integration, and Unit Test Plan, and the Validation Test Plan

2.5 Testing and Verification Tools

EMgine's development uses the C# programming language because it is one of the languages supported in Unity, a well-known game development platform (Unity Technologies, 2022b). The supporting Integrated Development Environment (IDE), Microsoft Visual Studio (VS), is the default script editor in Unity. EMgine development uses VS 2022 (Community Edition), which can access the following tools:

- **NUnit Unit Testing Framework**
  This supports the bulk of the automated testing approach for unit, integration, system, and regression testing. The IDE is configured to automatically run existing unit tests when it is compiling the code base. Unity Testing Framework uses custom integration of NUnit 3.5 (Unity Technologies, 2022c).
- **Moq Library for .NET**
  This supports tests that rely on components that do not have a concrete implementation, such as the user-defined data types It allows the definition of mocked interface calls within unit tests that are type-safe (Moq, 2022).
- **Performance Analysis**
  EMgine uses the performance tools built into VS 2022, which includes CPU, memory, and time usage tools (Jones et al., 2022).
- **Code Style and Quality Analyzers**
  EMgine's development uses the official .NET Compiler Platform (Roslyn) (.NET Platform, 2021) and the third-party Roslynator (Pihrt, 2022) analyzers to help adhere to good code quality and style practices. The Unity documentation also references Roslyn analyzers for code style and quality (Unity Technologies, 2022a).

3 System and Integration Test Description

These tests evaluate EMgine's implementation for adherence to the solution described in the Software Requirements Specification (SRS). Dependencies between data types and methods means that some tests are also integration-level tests.

3.1 Tests for Functional Requirements

EMgine's SRS clearly distinguishes between its data types and methods. Therefore, the test plan separates tests into groups for EMgine-defined data types (Section 3.1.1) and methods (Section 3.1.2).

3.1.1 EMgine-Defined Data Types

These tests evaluate the correctness and precision of EMgine-defined data types as described in the SRS. Tests that check for adherence to data type constraints (R1) also address EMgine's response to recoverable (Section 3.2.1) errors. Unless otherwise specified:

- The NUnit Unit Testing Framework (Section 2.5) automates all tests,
- Tests have an error tolerance $\epsilon = 1 \times 10^{-15}$, and
- All user messages are printed to the console.

Section 4.1 describes tests of other functions necessary to manipulate data types that are *not* described in the SRS.

3.1.1.1 Emotion Intensity $\mathbb{I}$ and Intensity Change $\mathbb{I}_\Delta$ These tests check the satisfaction of R2, R3, and adherence to the constraints on $\mathbb{I}$ and $\mathbb{I}_\Delta$ (R1). This includes testing their constructors and comparison methods.

1. systemtest-IntensityConstructor_GivenPositiveNumber

Initial State	New Session
Input	$5 : \mathbb{R}$
Expected Output	$5 : \mathbb{I}$
User Message	–

2. systemtest-IntensityConstructor_GivenPositiveNumberWithDecimals-1

Initial State	New Session
Input	$5.8 : \mathbb{R}$
Expected Output	$5.8 : \mathbb{I}$
User Message	–

3. systemtest-IntensityConstructor_GivenPositiveNumberWithDecimals-2

Initial State	New Session
Input	$2.000000000000001 : \mathbb{R}$
Expected Output	$2.000000000000001 : \mathbb{I}$
User Message	–

4. systemtest-IntensityConstructor_GivenZero

Initial State	New Session
Input	$0 : \mathbb{R}$
Expected Output	$0 : \mathbb{I}$
User Message	–

5. systemtest-IntensityConstructor_GivenNegativeNumber

Initial State	New Session
Input	$-5 : \mathbb{R}$
Expected Output	$0 : \mathbb{I}$
Test Case Derivation	When given a negative value, EMgine sets the intensity value to 0 because values of $\mathbb{I}$ must be ≥ 0.
User Message	`Warning: Value for emotion intensity is out of bounds. Clamping to the range [0, infty).`

6. systemtest-IntensityChgConstructor_GivenPositiveNumber

Initial State	New Session
Input	$5 : \mathbb{R}$
Expected Output	$5 : \mathbb{I}_{\Delta}$
User Message	–

7. systemtest-IntensityChgConstructor_GivenPositiveNumberWithDecimals-1

Initial State	New Session
Input	$5.8 : \mathbb{R}$
Expected Output	$5.8 : \mathbb{I}_{\Delta}$
User Message	–

8. systemtest-IntensityChgConstructor_GivenPositiveNumberWithDecimals-2

Initial State	New Session
Input	$2.000000000000001 : \mathbb{R}$
Expected Output	$2.000000000000001 : \mathbb{I}_{\Delta}$
User Message	–

9. systemtest-IntensityChgConstructor_GivenZero

Initial State	New Session
Input	$0 : \mathbb{R}$
Expected Output	$0 : \mathbb{I}_{\Delta}$
User Message	–

10. systemtest-IntensityChgConstructor_GivenNegativeNumber

Initial State	New Session
Input	$-5 : \mathbb{R}$
Expected Output	$-5 : \mathbb{I}_{\Delta}$
User Message	–

11. systemtest-IntensityChgConstructor_GivenNegativeNumberWithDecimals

Initial State	New Session
Input	$-1.000000000000007 : \mathbb{R}$
Expected Output	$-1.000000000000007 : \mathbb{I}_{\Delta}$
User Message	–

12. systemtest-CompareToIntensity_FirstIsLarger-1

Initial State	$i1 : \mathbb{I} = 5$, $i2 : \mathbb{I} = 1$
Input	$i1$.CompareToIntensity($i2$)
Expected Output	$1 : \mathbb{Z}$
User Message	–

13. systemtest-CompareToIntensity_FirstIsLarger-2

Initial State	$i1 : \mathbb{I} = 5$, $i2 : \mathbb{I} = 2.1$
Input	$i1$.CompareToIntensity($i2$)
Expected Output	$1 : \mathbb{Z}$
User Message	–

14. systemtest-CompareToIntensity_FirstIsSmaller-1

Initial State	$i1 : \mathbb{I} = 1$, $i2 : \mathbb{I} = 5$
Input	$i1$.CompareToIntensity($i2$)
Expected Output	$-1 : \mathbb{Z}$
User Message	–

15. systemtest-CompareToIntensity_FirstIsSmaller-2

Initial State	$i1 : \mathbb{I} = 5$, $i2 : \mathbb{I} = 5.8$
Input	$i1$.CompareToIntensity($i2$)
Expected Output	$-1 : \mathbb{Z}$
User Message	–

16. systemtest-CompareToIntensity_FirstIsSmaller-3

Initial State	$i1 : \mathbb{I} = 2$, $i2 : \mathbb{I} = 2.000000000000001$
Input	$i1$.CompareToIntensity($i2$)
Expected Output	$-1 : \mathbb{Z}$
User Message	–

17. systemtest-CompareToIntensity_EqualValues

Initial State	$i1 : \mathbb{I} = 5$, $i2 : \mathbb{I} = 5$
Input	$i1$.CompareToIntensity($i2$)
Expected Output	$0 : \mathbb{Z}$
User Message	–

18. systemtest-EqualsMinIntensity_IntensityIsLarger-1

Initial State	$i : \mathbb{I} = 5$, $i_{min} : \mathbb{I} = 0$
Input	i.EqualsMinIntensity()
Expected Output	$False : \mathbb{B}$
User Message	–

19. systemtest-EqualsMinIntensity_IntensityIsLarger-2

Initial State	$i : \mathbb{I} = 2.3$, $i_{min} : \mathbb{I} = 0$
Input	i.EqualsMinIntensity()
Expected Output	$False : \mathbb{B}$
User Message	–

20. systemtest-EqualsMinIntensity_IntensityIsLarger-3

Initial State	$i : \mathbb{I} = 0.000000000000001$, $i_{min} : \mathbb{I} = 0$
Input	i.EqualsMinIntensity()
Expected Output	$False : \mathbb{B}$
User Message	–

21. systemtest-EqualsMinIntensity_IntensityIsMin

Initial State	$i : \mathbb{I} = 0$, $i_{min} : \mathbb{I} = 0$
Input	i.EqualsMinIntensity()
Expected Output	$True : \mathbb{B}$
User Message	–

22. systemtest-CompareToIntensityChg_FirstIsLarger-1

Initial State	$d1 : \mathbb{I}_\Delta = 5$, $d2 : \mathbb{I}_\Delta = 0$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$1 : \mathbb{Z}$
User Message	–

23. systemtest-CompareToIntensityChg_FirstIsLarger-2

Initial State	$d1 : \mathbb{I}_\Delta = 5.8$, $d2 : \mathbb{I}_\Delta = 2.1$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$1 : \mathbb{Z}$
User Message	–

24. systemtest-CompareToIntensityChg_FirstIsLarger-3

Initial State	$d1 : \mathbb{I}_\Delta = 5.8$, $d2 : \mathbb{I}_\Delta = -1.7$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$1 : \mathbb{Z}$
User Message	–

25. systemtest-CompareToIntensityChg_FirstIsSmaller-1

Initial State	$d1 : \mathbb{I}_\Delta = 0$, $d2 : \mathbb{I}_\Delta = 5$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$-1 : \mathbb{Z}$
User Message	–

26. systemtest-CompareToIntensityChg_FirstIsSmaller-2

Initial State	$d1 : \mathbb{I}_\Delta = 5$, $d2 : \mathbb{I}_\Delta = 5.8$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$-1 : \mathbb{Z}$
User Message	–

27. systemtest-CompareToIntensityChg_FirstIsSmaller-3

Initial State	$d1 : \mathbb{I}_\Delta = -1.7$, $d2 : \mathbb{I}_\Delta = 5.8$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$-1 : \mathbb{Z}$
User Message	–

28. systemtest-CompareToIntensityChg_FirstIsSmaller-4

Initial State	$d1 : \mathbb{I}_\Delta = -1.7$, $d2 : \mathbb{I}_\Delta = 0$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$-1 : \mathbb{Z}$
User Message	–

29. systemtest-CompareToIntensityChg_FirstIsSmaller-5

Initial State	$d1 : \mathbb{I}_\Delta = 5$, $d2 : \mathbb{I}_\Delta = 5.000000000000001$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$-1 : \mathbb{Z}$
User Message	–

30. systemtest-CompareIntensityChg_EqualValues

Initial State	$d1 : \mathbb{I}_\Delta = 5$, $d2 : \mathbb{I}_\Delta = 5$
Input	$d1$.CompareToIntensityChg($d2$)
Expected Output	$0 : \mathbb{Z}$
User Message	–

31. systemtest-CompareIntensityChgtoMinIntensity_ChgIsLarger-1

Initial State	$d : \mathbb{I}_\Delta = 5$, $i_{min} : \mathbb{I} = 0$
Input	d.CompareToMinIntensity()
Expected Output	$1 : \mathbb{Z}$
User Message	–

32. systemtest-CompareIntensityChgtoMinIntensity_ChgIsLarger-2

Initial State	$d : \mathbb{I}_\Delta = 5.8$, $i_{min} : \mathbb{I} = 0$
Input	d.CompareToMinIntensity()
Expected Output	$1 : \mathbb{Z}$
User Message	–

33. systemtest-CompareIntensityChgtoMinIntensity_ChgIsLarger-3

Initial State	$d : \mathbb{I}_\Delta = 2.1$, $i_{min} : \mathbb{I} = 0$
Input	d.CompareToMinIntensity()
Expected Output	$1 : \mathbb{Z}$
User Message	–

34. systemtest-CompareIntensityChgtoMinIntensity_ChgIsSmaller-1

Initial State	$d : \mathbb{I}_\Delta = -5$, $i_{min} : \mathbb{I} = 0$
Input	d.CompareToMinIntensity()
Expected Output	$-1 : \mathbb{Z}$
User Message	–

35. systemtest-CompareIntensityChgtoMinIntensity_ChgIsSmaller-2

Initial State	$d : \mathbb{I}_\Delta = -1.7$, $i_{min} : \mathbb{I} = 0$
Input	d.CompareToMinIntensity()
Expected Output	$-1 : \mathbb{Z}$
User Message	–

36. systemtest-CompareIntensityChgtoMinIntensity_ChgIsSmaller-3

Initial State	$d : \mathbb{I}_\Delta = -0.000000000000001$, $i_{min} : \mathbb{I} = 0$
Input	d.CompareToMinIntensity()
Expected Output	$-1 : \mathbb{Z}$
User Message	–

37. systemtest-CompareIntensityChgtoMinIntensity_ChgIsEqual

Initial State	$d : \mathbb{I}_\Delta = 0$, $i_{min} : \mathbb{I} = 0$
Input	d.CompareToMinIntensity()
Expected Output	$0 : \mathbb{Z}$
User Message	–

3.1.2 EMgine Methods

These tests evaluate the correctness and precision of EMgine's methods as described in the SRS. Unless otherwise specified:

- The NUnit Unit Testing Framework (Section 2.5) automates all tests,
- Tests have an error tolerance $\epsilon = 1 \times 10^{-15}$, and
- All user messages are printed to the console.

3.1.2.1 Manipulating Emotion Intensities These tests check the satisfaction of R27.

1. systemtest-UpdateWithChg_GivenPositiveChg-1

Initial State	$i : \mathbb{I} = 2,\ d : \mathbb{I}_\Delta = 1$
Input	i.UpdateWithChg(d)
Expected Output	$i' = 0.1 \cdot \log_2(2^{20} + 2^{10})$
User Message	–

2. systemtest-UpdateWithChg_GivenPositiveChg-2

Initial State	$i : \mathbb{I} = 40.89,\ d : \mathbb{I}_\Delta = 1$
Input	i.UpdateWithChg(d)
Expected Output	$i' = 0.1 \cdot \log_2(2^{408.9} + 2^{10})$
User Message	–

3. systemtest-UpdateWithChg_GivenNegativeChg-1

Initial State	$i : \mathbb{I} = 2,\ d : \mathbb{I}_\Delta = -1$
Input	i.UpdateWithChg(d)
Expected Output	$i' = 0.1 \cdot \log_2(2^{20} - 2^{10})$
User Message	–

4. systemtest-UpdateWithChg_GivenNegativeChg-2

Initial State	$i : \mathbb{I} = 40.89,\ d : \mathbb{I}_\Delta = -1$
Input	i.UpdateWithChg(d)
Expected Output	$i' = 0.1 \cdot \log_2(2^{408.9} - 2^{10})$
User Message	–

5. systemtest-UpdateWithChg_GivenZeroChg-1

Initial State	$i : \mathbb{I} = 2,\ d : \mathbb{I}_\Delta = 0$
Input	i.UpdateWithChg(d)
Expected Output	$i' = i$
User Message	–

6. systemtest-UpdateWithChg_GivenZeroChg-2

Initial State	$i : \mathbb{I} = 40.89,\ d : \mathbb{I}_\Delta = 0$
Input	i.UpdateWithChg(d)
Expected Output	$i' = i$
User Message	–

7. systemtest-UpdateWithChg_IntensityZero

Initial State	$i : \mathbb{I} = 0,\ d : \mathbb{I}_\Delta = 1$
Input	i.UpdateWithChg(d)
Expected Output	$i' = 0.1 \cdot \log_2(1 + 2^{10})$
User Message	–

3.2 Tests for Nonfunctional Requirements

These tests evaluate the ability for EMgine's implementation to support the Robustness and Performance nonfunctional requirement categories in EMgine's SRS, as well as NF6 from the Verifiability category, because they depend on EMgine's implementation and test organization alone.

Nonfunctional requirement NF7 and those in the categories of Maintainability, Reusability, Portability, Operational & Interoperability, Understandability, Usability, Look & Feel, Culture (World), Culture (Game Development Work), and Legal are not tested here because they require end-user involvement. This is addressed in EMgine's Acceptance Test Plan (ATP).

3.2.1 Recoverable User Error

EMgine relies on warning messages, not error messages, to inform users when it encounters a recoverable error (NF1). It must also produce a valid output to compensate for its inability to produce one from user-provided inputs. Relevant tests for EMgine-defined data types (Section 3.1.1) are:

- systemtest-IntensityConstructor_GivenNegativeNumber

3.2.2 Atomic Test Units

EMgine's architecture suggests that it can be verified on a per-component basis. Therefore, EMgine's test suite must demonstrate that the tests related to each component can be run independently of the others (NF6). Inputs required from other components must be mocked to demonstrate the component's ability to operate as expected to build confidence in its ability to integrate with non-EMgine methods.

3.3 Traceability Between Test Cases and Requirements

In the traceability graphs, test suites appear at the tail of an arrow and requirements appear at the head. In the traceability matrices, the test suites related to a requirement's verification are marked with an "X" in the requirement's column.

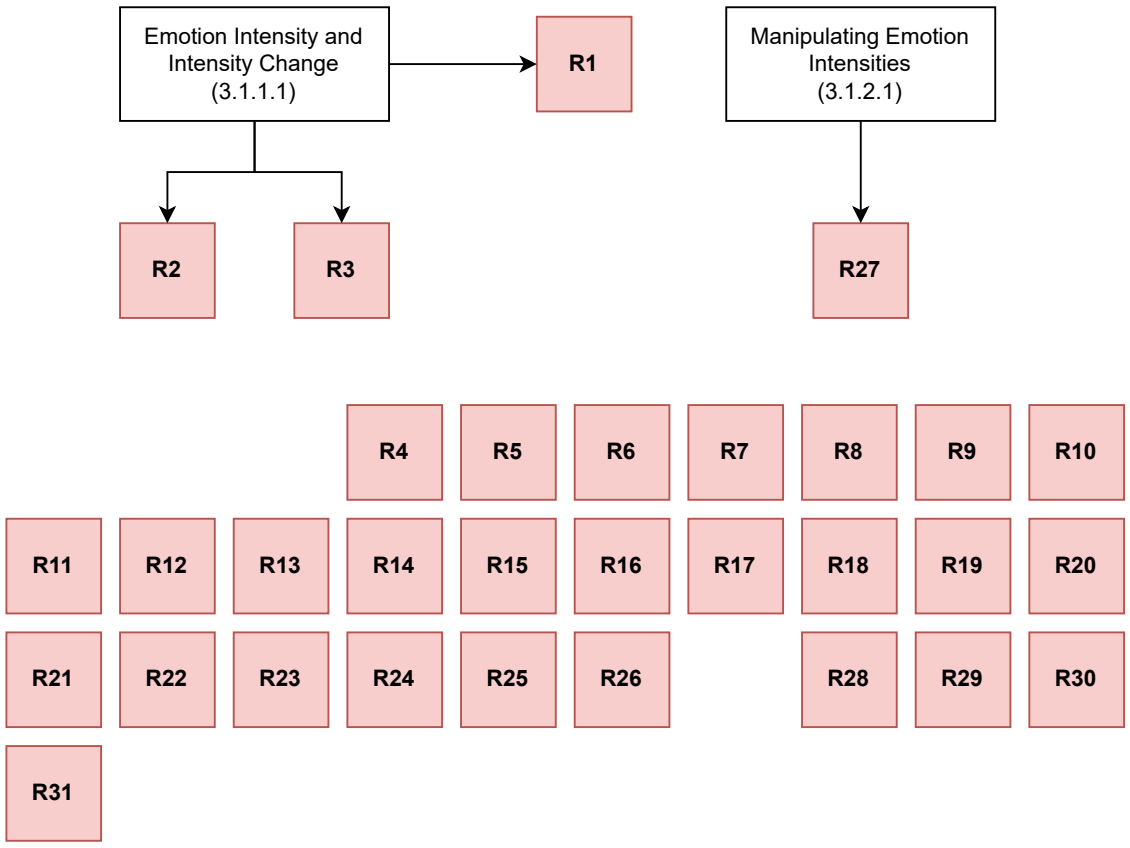

Figure 2: Traceability between System and Integration-Level Test Suites and Functional Requirements

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31
3.1.1.1	X	X	X																												
3.1.2.1																											X				

Table 1: Traceability between System and Integration-Level Test Suites and Functional Requirements

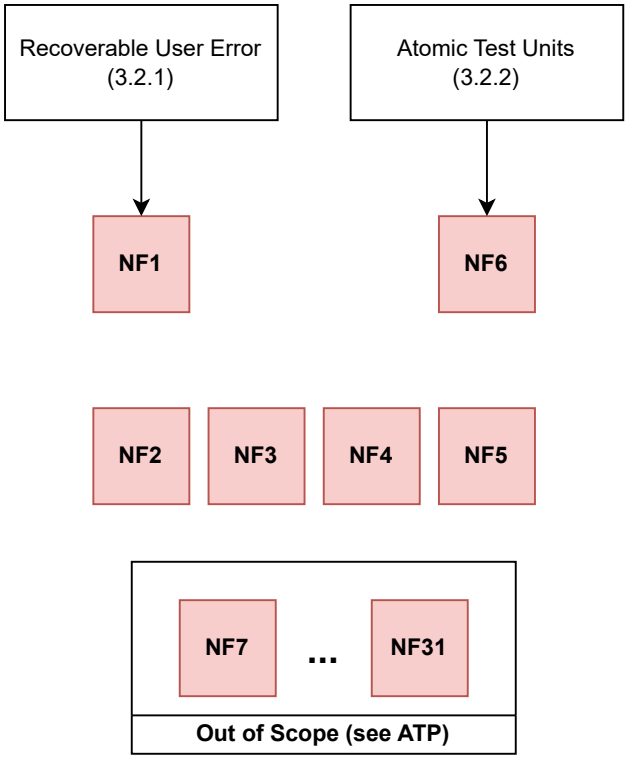

Figure 3: Traceability between System and Integration-Level Test Suites and Nonfunctional Requirements

	NF1	NF2	NF3	NF4	NF5	NF6
3.2.1	X					
3.2.2						X

Table 2: Traceability between System and Integration-Level Test Suites and Nonfunctional Requirements

4 Unit Test Description

These tests exercise all functionality described in EMgine's Module Interface Specification (Version 0.1.1) not covered by the system-level tests (Section 3). They are organized by module.

4.1 Tests for Functional Requirements

Unless otherwise specified:

- The NUnit Unit Testing Framework (Section 2.5) automates all tests,
- Tests with $\mathbb{R}$ return types have an error tolerance $\epsilon = 1 \times 10^{-15}$, and
- All user messages are printed to the console.

4.1.1 Emotion Intensity Types Module (M1)

1. unittest-Normalize_GivenPositiveIntensity-1

Initial State	$i1 : \mathbb{I} = 5$, $i2 : \mathbb{I} = 2.5$
Input	$i1$.Normalize($i2$)
Expected Output	$2 : \mathbb{I}$
User Message	–

2. unittest-Normalize_GivenPositiveIntensity-2

Initial State	$i1 : \mathbb{I} = 5$, $i2 : \mathbb{I} = 1.6$
Input	$i1$.Normalize($i2$)
Expected Output	$3.125 : \mathbb{I}$
User Message	–

3. unittest-Normalize_GivenPositiveIntensity-3

Initial State	$i1 : \mathbb{I} = 2.5$, $i2 : \mathbb{I} = 5$
Input	$i1$.Normalize($i2$)
Expected Output	$0.5 : \mathbb{I}$
User Message	–

4. unittest-Normalize_GivenPositiveIntensity-4

Initial State	$i1 : \mathbb{I} = 2.5$, $i2 : \mathbb{I} = 1.6$
Input	$i1$.Normalize($i2$)
Expected Output	$1.5625 : \mathbb{I}$
User Message	–

5. unittest-Normalize_GivenPositiveIntensity-5

Initial State	$i1 : \mathbb{I} = 2.5$, $i2 : \mathbb{I} = 2.5$
Input	$i1$.Normalize($i2$)
Expected Output	$1 : \mathbb{I}$
User Message	–

6. unittest-Normalize_GivenZeroIntensity

Initial State	$i1 : \mathbb{I} = 5$, $i2 : \mathbb{I} = 0$
Input	$i1$.Normalize($i2$)
Expected Output	`null`
Test Case Derivation	Normalization is division-based and divide-by-zero is undefined. EMgine cannot normalize an intensity when the normalization factor is zero.
User Message	`Error: Reference intensity has a value of zero. Cannot complete normalization.`

7. unittest-Normalize_IntensityToNormalizeIsZero

Initial State	$i1 : \mathbb{I} = 0$, $i2 : \mathbb{I} = 1.6$
Input	$i1$.Normalize($i2$)
Expected Output	$0 : \mathbb{I}$
User Message	–

8. unittest-ScaleByValue_GivenPositiveValue-1

Initial State	$d : \mathbb{I}_\Delta = 0$, $v : \mathbb{R} = 4.5$
Input	d.ScaleByValue(v)
Expected Output	$0 : \mathbb{I}_\Delta$
User Message	–

9. unittest-ScaleByValue_GivenPositiveValue-2

Initial State	$d : \mathbb{I}_\Delta = -1.5$, $v : \mathbb{R} = 4.5$
Input	d.ScaleByValue(v)
Expected Output	$-6.75 : \mathbb{I}_\Delta$
User Message	–

10. unittest-ScaleByValue_GivenPositiveValue-3

Initial State	$d : \mathbb{I}_\Delta = 2.8$, $v : \mathbb{R} = 4.5$
Input	d.ScaleByValue(v)
Expected Output	$12.6 : \mathbb{I}_\Delta$
User Message	–

11. unittest-ScaleByValue_GivenNegativeValue-1

Initial State	$d : \mathbb{I}_\Delta = 0$, $v : \mathbb{R} = -1$
Input	d.ScaleByValue(v)
Expected Output	$0 : \mathbb{I}_\Delta$
User Message	–

12. unittest-ScaleByValue_GivenNegativeValue-2

Initial State	$d : \mathbb{I}_\Delta = -1.5$, $v : \mathbb{R} = -1$
Input	d.ScaleByValue(v)
Expected Output	$1.5 : \mathbb{I}_\Delta$
User Message	–

13. unittest-ScaleByValue_GivenNegativeValue-3

Initial State	$d : \mathbb{I}_\Delta = 2.8$, $v : \mathbb{R} = -1$
Input	d.ScaleByValue(v)
Expected Output	$-2.8 : \mathbb{I}_\Delta$
User Message	–

14. unittest-ScaleByValue_GivenZero-1

Initial State	$d : \mathbb{I}_\Delta = 0$, $v : \mathbb{R} = 0$
Input	d.ScaleByValue(v)
Expected Output	$0 : \mathbb{I}_\Delta$
User Message	–

15. unittest-ScaleByValue_GivenZero-2

Initial State	$d : \mathbb{I}_\Delta = -1.5$, $v : \mathbb{R} = 0$
Input	d.ScaleByValue(v)
Expected Output	$0 : \mathbb{I}_\Delta$
User Message	–

16. unittest-ScaleByValue_GivenZero-3

Initial State	$d : \mathbb{I}_\Delta = 2.8$, $v : \mathbb{R} = 0$
Input	d.ScaleByValue(v)
Expected Output	$0 : \mathbb{I}_\Delta$
User Message	–

17. unittest-ToReal_Intensity-1

Initial State	$i : \mathbb{I} = 0$
Input	i.ConvertToValue()
Expected Output	$0 : \mathbb{R}$
User Message	–

18. unittest-ToReal_Intensity-2

Initial State	$i : \mathbb{I} = 2.3$
Input	i.ConvertToValue()
Expected Output	$2.3 : \mathbb{R}$
User Message	–

19. unittest-ToReal_Intensity-3

Initial State	$i : \mathbb{I} = 5$
Input	i.ConvertToValue()
Expected Output	$5 : \mathbb{R}$
User Message	–

20. unittest-ToReal_IntensityChg-1

Initial State	$d : \mathbb{I}_\Delta = 0$
Input	d.ConvertToValue()
Expected Output	$0 : \mathbb{R}$
User Message	–

21. unittest-ToReal_IntensityChg-2

Initial State	$d : \mathbb{I}_\Delta = -2.3$
Input	d.ConvertToValue()
Expected Output	$-2.3 : \mathbb{R}$
User Message	–

22. unittest-ToReal_IntensityChg-3

Initial State	$d : \mathbb{I}_{\Delta} = 5$
Input	d.ConvertToValue()
Expected Output	$5 : \mathbb{R}$
User Message	–

4.2 Traceability Between Test Cases and Modules

In the traceability graphs, test suites appear at the tail of an arrow and modules appear at the head. In the traceability matrices, the test suites related to a module's verification are marked with an "X" in the requirement's column.

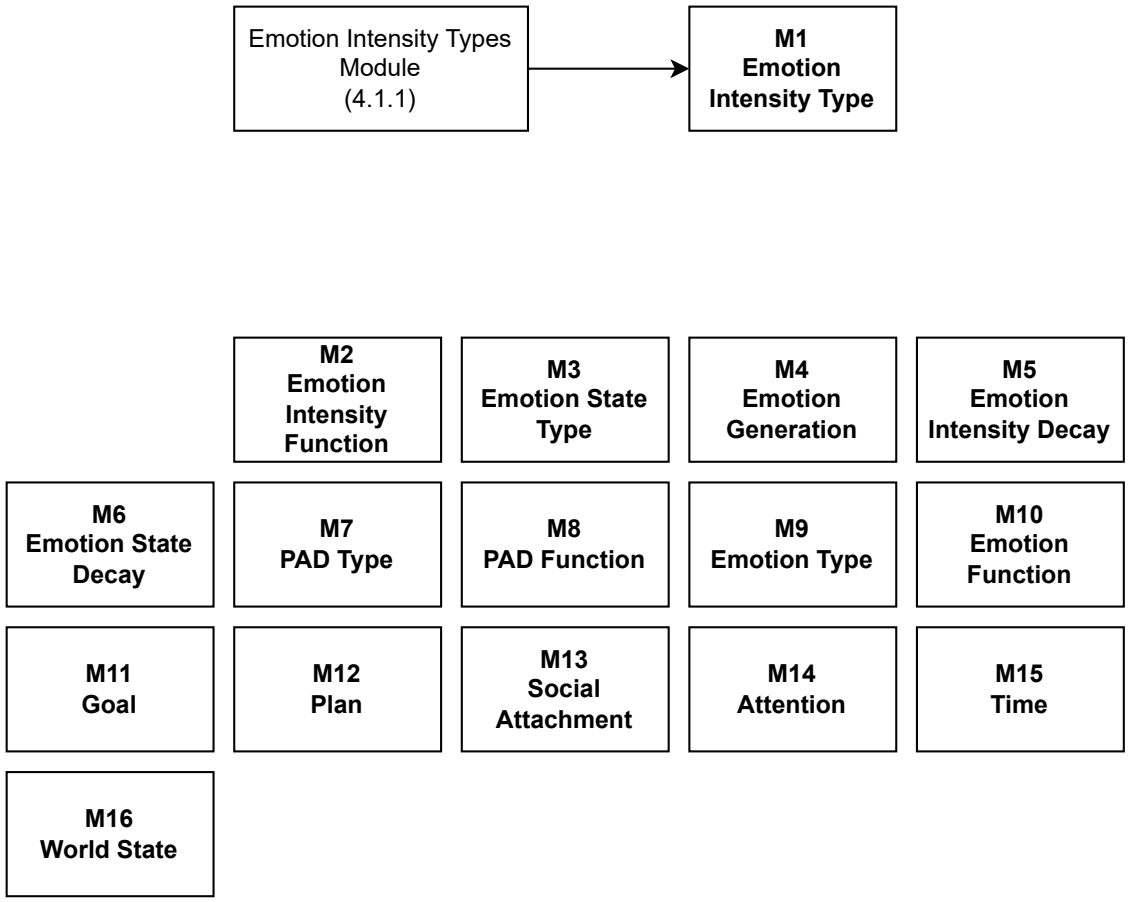

Figure 4: Traceability between Unit-Level Test Suites and Modules

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
4.1.1	X															

Table 3: Traceability between Unit-Level Test Suites and Modules

5 References

Joost Broekens, Eva Hudlicka, and Rafael Bidarra. 2016. Emotional Appraisal Engines for Games. In *Emotion in Games: Theory and Praxis*, Kostas Karpouzis and Georgios N. Yannakakis (Eds.). Socio-Affective Computing, Vol. 4. Springer International Publishing, Cham, Switzerland, Chapter 13, 215–232. https://doi.org/10.1007/978-3-319-41316-7_13

IEEE Computer Society. 2008. *IEEE Standard for Software and System Test Documentation*. Technical Report IEEE Std 829-2008. IEEE, New York, NY, USA. https://doi.org/10.1109/IEEESTD.2008.4578383

IEEE Computer Society. 2009. *IEEE Recommended Practice for Software Requirements Specifications*. Technical Report IEEE Std 830-1998 (R2009). IEEE, New York, NY, USA. https://doi.org/10.1109/IEEESTD.1998.88286

IEEE Computer Society. 2017. *IEEE Standard for System, Software, and Hardware Verification and Validation*. Technical Report IEEE Std 1012-2016. IEEE, New York, NY, USA. https://doi.org/10.1109/IEEESTD.2017.8055462

ISO, IEC, and IEEE. 2015. *ISO/IEC/IEEE International Standard - Systems and software engineering – System life cycle processes*. Technical Report ISO/IEC/IEEE 15288:2015(E). IEEE, New York, NY, USA. https://doi.org/10.1109/IEEESTD.2015.7106435

Mike Jones, Misty Hays, Gordon Hogenson, J. Martens, David Coulter, Shannon Leavitt, Meg van Huygen, Clément Habinshuti, Genevieve Warren, Andy Sterland, Kraig Brockschmidt, Patricia McNary, Kristine Toliver, and Mark McGee. 2022. First look at profiling tools (C#, Visual Basic, C++, F#). Retrieved February 28, 2023 from https://learn.microsoft.com/en-us/visualstudio/profiling/profiling-feature-tour?view=vs-2022

Moq. 2022. Moq 4.x. Retrieved February 28, 2023 from https://github.com/moq/moq4

.NET Platform. 2021. Roslyn. Retrieved February 28, 2023 from https://github.com/dotnet/roslyn

Josef Pihrt. 2022. Roslynator. Retrieved February 28, 2023 from https://github.com/JosefPihrt/Roslynator

Unity Technologies. 2022a. Roslyn analyzers and source generators. Retrieved February 28, 2023 from https://docs.unity3d.com/Manual/roslyn-analyzers.html

Unity Technologies. 2022b. Setting Up Your Scripting Environment. Retrieved February 28, 2023 from https://docs.unity3d.com/Manual/ScriptingSettingUp.html

Unity Technologies. 2022c. Unity Test Framework overview. Retrieved February 28, 2023 from https://docs.unity3d.com/Packages/com.unity.test-framework@2.0/manual/index.html