By Ngaba Franck

MetaScript

# General overview of the Language

# Preambule

This document provides an overview of the project, including key objectives, methodologies, and anticipated outcomes.

# Chapter 1

# Context of MetaScript

*Let us imagine a scenario where the creation of graphical interfaces, video games, virtual or augmented reality experiences and even virtual simulations are not limited by conventional tools.*

*By Conventional tools, we mean technologies, frameworks or programming language usen nowadays.* ( ex : Three.Js pour du JavaScript, Unity pour du C# ), *all of them have some limitations, whether in terms of complexity, rigidity, or performance, which can hamper the creativity of developers and designers. MetaScript is there to turn this vision into reality.*

*It's not simply a new item for the UI developments. It's a new way of paradigm changement, created to redefine the standards of interactive and immersive design. this programming language allows you to push the boundaries of what is possible. whether creating intuitive interfaces, breathtaking games, or virtual and augmented reality environments of unprecedented depth, MetaScript is the ally that offers you speed of execution without compromising on quality.*

## 1.1 Virtual Simulation, What is it

### 1.1.1 Meta-World

Theorem 1. *A* meta world $W_0$ *is characterized by these properties:*

- $W_0$ *is made up of elements with which we can physically interact by* touching, seeing, listening, tasting or smelling
- $W_0$ *is made up of elements to which we have attributed a meaning. These elements are designated by* $w_{0,i}$ *with i ∈N*
- *The* $w_{0,i}$ *are splitted into 2 categories : the liven ones and the static ones*

Theorem 2. *The world where human beings are living is a Meta-World. It is called by convention the* real World *and it is designated by* $\Omega$W *and in this world, humans are an example on liven element.*

Theorem 3. *meta(W) is the set which gather all the meta-worlds*

Theorem 4. *Let us considerate 2 meta-worlds* $W_1$ *and* $W_2$. *Simulation is a set of finite functions {* $f_1$ $f_2$, ..., $f_n$ *} which assure that* $\forall w_1 \in W_1 \ \exists f_i$ , $\exists w_2 \in W_2$ , $f_i(w_1)$ *) =* $w_2$ *, i ∈{1,2,...,n*

All functions which define a Simulation must be finite, meaning that they are either clearly stated either it exists some rationnal expressions to check if a function is defined into that simulation.

### 1.1.2 Meta-World Component

Theorem 5. *Let us take a Meta-World* $W_0$. *Everything which are defined in this meta-world is its* meta-world component. $w_0 \in W_0$ *means that* $w_0$ *is a meta-world component of* $W_0$

- $w_0$ *is made up of elements with which we can physically interact by* touching, seeing, listening, tasting or smelling
- *a meta-world component is a set of differents characteristics.* $w_0$ *= {'attr1','attr2',...'attrn'}.*
  *For example a meta-world component can have a name, a size, a colour, a position ans so many others possibilities*

- *all meta-world component has at least one attribute, this attribute tell either or not the component is alive or not*
- *all meta-world component can have so many finite attributes as possibles, even another one meta-world component. However all the attributes of a meta-world component must be defined in the same meta-world as its.*

### 1.1.3 Meta-World Phenomenon

Theorem 6. *A Meta-World Phenomenon is some event that occurs into a meta-world which affect one or more of its components. we represent a phenomenon defined in a meta-World $W_0$ by this term $f_{W0}$.*

- *a phenomenon can affect a component either it is alive or not.*
- *a phenomenon just happens to components defined in the meta-eorld where it is defined*
- *a phenomenon is declenched by something, it can be a component, another phenomenon or the meta-world itself where they are contained*
- – *if a phenomenon $f_{W0}$. is launched by a component $w_0$, this function is written $w_0 f_{W0}$.*
  – *if a phenomenon $f_{W0}$. is launched by a meta-world itself, this function is just written $f_{W0}$.*
  – *if a phenomenon $f_{W0}$. is launched by another phenomenon $g_{W0}$, this function is written $\frac{\Delta f_{W0}}{\Delta g_{W0}}$ = g'$f_{W0}$.*

### 1.1.4 Virtual Simulation

Theorem 7. *ΔW is the Meta-World which have been created by an human. By Convention, it is called the* virtual World *ΔW. ΔW is the only Meta-World which respect this property : $\forall w_1 \in \Omega W$, $\exists w_2 \in \Delta W$, $f_i(w_1)$ = $w_2$*

Theorem 8. *Virtualisation means to pass from a meta-world $W_a \subseteq \Omega W$ to a meta-world $W_b \subseteq \Delta W$*

Theorem 9. *Virtual Creation means to pass from a meta-world $W_a \subseteq \Delta W$ to another meta-world $W_b \subseteq \Delta W$*

Theorem 10. *If $W_b \subseteq \Delta W$ is defined by virtualisation from another meta-world $W_a \subseteq \Omega W$.*
*We call the real-world experimentation, thes simulation which means to pass from $W_b$ to $W_a$.*

### 1.1.5 Some Examples of Simulations

## 1.2 Types of Virtual Simulations made by Computer

Theorem 11. *we call Simulating Setup all the set of components which in charge to make a simulation*

There are many kinds of simulations, every process where we are corresponding some World elements to another one can be considered as a simulation.
Imagination for example is a meta-world, that is its formal definition.

- starting meta-world : *some scene that a people has already lived*
- simulating setup : *human mind*
- ending meta-world : *a mind-created world where static items are represented as the setup"s memory and events occurs as the setup remembered them or wanted them to happens.*

In our case, we still focus in Virtual Simulations where the setup is a computer

### 1.2.1 2D-World Simulation

*2D Simulation is one of the lowest exigence capacities in simulations* a 2D world is a word where the space has just 2 axis, one for the width and one another for the height. ( An example is a paper sheet )

### 1.2.2 3D-World Simulation

*3D Simulation is* a 3D World is

### 1.2.3 Mathematical Models Simulation

This simulation is kinda different of others virtuals ones. It returns mainly

### 1.2.4 The Case of Simulation made by IA

This simulation is kinda different of others virtuals ones. It returns mainly

## 1.3 Problems solved with Virtual Simulations

Virtual environments, such as immersive simulations and 3D virtual worlds, offer a multitude of solutions for various problems in several fields. Virtual simulations are powerful tools used across diverse domains to solve problems related to training, design, testing, risk assessment, decision-making, and optimization. By providing a safe, cost-effective, and flexible platform, they enable innovation, enhance safety, reduce costs, and improve efficiency in various industries and fields.:

### 1.3.1 HealthCare and Medicine

Applications

- Surgical Training: Surgeons can practice complex procedures in a virtual environment, reducing the risk of errors during actual surgeries.
- Medical Diagnosis and Treatment: Simulations can help doctors understand the effects of diseases on the body and predict outcomes of different treatments.
- Healthcare Planning: Simulations can model patient flow in hospitals to optimize resource allocation and improve patient care.

Problems Solved

- Reducing risks associated with medical training.
- Improving diagnosis and treatment planning.
- Enhancing patient safety and operational efficiency

### 1.3.2 Aerospace and Aviation

Applications

- Flight Training: Pilots use flight simulators to practice maneuvers, emergency procedures, and instrument navigation without real-world risks.
- Aircraft Design and Testing: Simulations are used to test aerodynamics, structural integrity, and system performance of new aircraft designs.
- Air Traffic Management: Simulations help in designing and optimizing air traffic control systems for safer and more efficient air travel.

Problems Solved

- Reducing costs and risks associated with pilot training.
- Improving aircraft safety and performance.
- Enhancing air traffic management and reducing congestion.

### 1.3.3 Military and Defense

Applications

- Combat Training: Virtual simulations provide a realistic environment for soldiers to practice combat scenarios, tactics, and strategy without real-world risks.
- Mission Planning: Simulations help in planning military operations by modeling potential outcomes and assessing risks.
- Equipment Testing: Air Traffic Management:Simulations are used to test new weapons, vehicles, and technologies in various combat scenarios.

Problems Solved

- Improving training effectiveness and readiness.
- Enhancing mission planning and execution.
- Reducing costs and risks associated with real-world testing.

### 1.3.4 Automotive Industry

Applications

- Vehicle Design and Testing: Simulations help engineers test vehicle performance, safety features, and aerodynamics without the need for physical prototypes.
- Driver Training: Virtual simulators are used to train drivers in safe driving techniques and to prepare them for various road conditions..
- Autonomous Vehicle Development: Simulations are crucial for testing and refining algorithms for self-driving cars.

Problems Solved

- Reducing costs and time associated with vehicle prototyping and testing.
- Improving driver safety and training.
- Accelerating the development of autonomous vehicles.

### 1.3.5 Engineering and Manufacturing

Applications

- Product Development: Virtual simulations allow for rapid prototyping, testing, and optimization of new products, reducing time-to-market.
- Process Optimization: Simulations help optimize manufacturing processes, improve efficiency, and reduce waste.
- Failure Analysis: Simulations are used to analyze potential failure points in products and processes, enhancing quality and reliability.

Problems Solved

- Reducing costs and time in product development.
- Improving manufacturing efficiency and reducing waste.
- Enhancing product quality and reliability.

### 1.3.6 Education and Training

Applications

- Students can conduct experiments in a virtual environment, enhancing understanding without the need for physical resources.
- Distance Learning: Simulations provide interactive, immersive learning experiences for remote learners.
- Professional Training: Professionals in fields like engineering, healthcare, and emergency response use simulations for hands-on training.

Problems Solved

- Enhancing learning experiences and engagement.
- Providing access to practical training without physical constraints.
- Reducing costs associated with physical labs and training facilities.

### 1.3.7 Urban Planning and Architecture

Applications

- City Planning: Simulations model traffic flow, population growth, and infrastructure needs to optimize urban development.
- Building Design: Virtual simulations help architects visualize and optimize building designs for aesthetics, functionality, and energy efficiency.
- Disaster Preparedness: Simulations model the impact of natural disasters on urban areas to improve emergency response and resilience.

Problems Solved

- Improving urban planning and infrastructure development.
- Enhancing building design and sustainability.
- Increasing preparedness for natural disasters and emergencies.

### 1.3.8 Entertainment and Gaming

Applications

- Game Development: Simulations create realistic environments and physics in video games, enhancing player experience.
- Virtual Reality (VR) and Augmented Reality (AR): Simulations are used to create immersive experiences for entertainment, education, and training.

Problems Solved

- Enhancing user engagement and experience in gaming.
- Providing new forms of interactive and immersive entertainment.
- Expanding the possibilities for storytelling and experiential learning.

### 1.3.9 Environmental Science and Climate Modeling

Applications

- Climate Modeling: Simulations predict the impact of various factors on climate change, helping in policy-making and environmental protection.
- Ecosystem Management: Simulations model the impact of human activities on ecosystems to develop sustainable management practices.
- Natural Resource Management: Virtual models help in managing resources like water, forests, and fisheries by predicting future scenarios.

Problems Solved

- Predicting and mitigating the impact of climate change.
- Promoting sustainable practices in ecosystem and resource management.
- Enhancing decision-making for environmental policies.

### 1.3.10 Finance and Economics

Applications

- Climate Modeling: Simulations predict the impact of various factors on climate change, helping in policy-making and environmental protection.
- Ecosystem Management: Simulations model the impact of human activities on ecosystems to develop sustainable management practices.
- Natural Resource Management: Virtual models help in managing resources like water, forests, and fisheries by predicting future scenarios.

Problems Solved

- Predicting and mitigating the impact of climate change.
- Promoting sustainable practices in ecosystem and resource management.
- Enhancing decision-making for environmental policies.

### 1.3.11 Telecommunications and Network Management

Applications

- Climate Modeling: Simulations predict the impact of various factors on climate change, helping in policy-making and environmental protection.
- Ecosystem Management: Simulations model the impact of human activities on ecosystems to develop sustainable management practices.
- Natural Resource Management: Virtual models help in managing resources like water, forests, and fisheries by predicting future scenarios.

Problems Solved

- Predicting and mitigating the impact of climate change.
- Promoting sustainable practices in ecosystem and resource management.
- Enhancing decision-making for environmental policies.

### 1.3.12 Construction and Civil Engineering

Applications

- Structural Analysis: Simulations assess the stability and durability of buildings, bridges, and other infrastructure under various conditions.
- Project Planning: Simulations optimize construction processes and resource allocation.
- Safety Training: Virtual environments provide construction workers with training on safety protocols and equipment handling.

Problems Solved

- Reducing risks and improving safety in construction.
- Enhancing project planning and efficiency.
- Improving structural integrity and durability of infrastructure.

## 1.4 Some of the actual Virtualizing tools

### 1.4.1 Game Engine

### 1.4.2 Dedicated Simulation Softwares

### 1.4.3 3D Modeling Tools

### 1.4.4 Virtual Reality and Augmented Reality Softwares

### 1.4.5 Educational Simulation and Training Tools

### 1.4.6 Cloud System Simulations

## 1.5 MetaScript, the Virtual Simulations Language

### 1.5.1 Limitations of the actuals technologies

### 1.5.2 MetaScript, the language made to assure world simulations

Chapter 2

# Presentation of the Langage

## 2.1 MetaScript Philosophy

Le MetaScript s'inspire essentiellement du C++. On peut le voir comme le C++ auquel on a ajoute une couche pour la simulation virtuelle. Toutefois la Syntaxe differe un peu avec le langage.

## 2.2 MetaScript Softwares Architecture

Les programmes concus en MetaScript sont subdivises en 3 entites :

- Un Cerveau , ou se trouve toute la partie logique du programme
- des Scenes qui sont des interfaces graphiques charges d'afficher le resultat des programmes
- un Connecteur entre le Cerveau et les Scenes qui se charge de transcrire les instructions en elements graphiques

Voici une illusration a l'aide de ce code :

```
        Load execLib;

        Program calcul1 {
            int number1,number2,result;
            char operator;
            connector scene_number[3];
            scene_number[0] = this.MAIN_SCREEN.readBox(&nombre1,"Prompt a First Number");
            scene_number[1] = this.MAIN_SCREEN.readBox(&nombre2,"Prompt a Second Number");
            scene_number[2] = this.MAIN_SCREEN.readChoicesBox(&operator,["+","-","x","/"])
                ;
            for ( int i=0; i<3: i++) {
                scene_number[i].changeEvt() => {
                    result = execLib.eval("{} {} {}".scene_number[0],scene_number[2],
                        scene_number[1]);
                }
            }
            this.MAIN_SCREEN.printBox(result);


         }
```

## 2.3 Caracteristics of MetaScript

That's some Caracteristics of the MetaScript

### 2.3.1 Expressivity

The syntax of Metascript is simple and intuitive, it herits from the C++ syntaxe. We want everyone able to acces to this laguage. Skillfully Creators to Pures novices.

### 2.3.2 Paradigm

2. Orienté objet et support d'objets virtuels

Pourquoi ? Les environnements virtuels sont naturellement constitués d'objets distincts (entités, avatars, objets physiques), ce qui fait de la programmation orientée objet un choix adapté. Conseil : MetaScript devrait avoir un système d'objets robuste avec des fonctionnalités comme l'héritage, la composition, et des abstractions adaptées aux environnements virtuels. Exemple : Des primitives comme Object3D, Entity, Agent pour modéliser des objets dans l'environnement avec des méthodes natives pour gérer les transformations 3D, la physique et les interactions.

### 2.3.3 2D & 3D Functions

3. Gestion intégrée de la 3D et des simulations physiques

Pourquoi ? La création d'environnements virtuels requiert la manipulation facile de la 3D et des simulations physiques (gravité, collisions, etc.). Conseil : Intégrer des primitives spécifiques pour la manipulation de la géométrie 3D, des transformations (rotation, translation, échelle), ainsi que des simulations physiques (moteur physique intégré pour gérer la gravité, les collisions, la dynamique). Exemple : Des fonctions comme applyGravity(), detectCollision(), ou simulatePhysics(timeStep) devraient être disponibles nativement.

### 2.3.4 Parallelism

4. Concurrence et Parallélisme pour la gestion des simulations

Pourquoi ? Les simulations complexes (comme les simulations physiques ou des mondes peuplés d'agents) peuvent nécessiter un traitement en parallèle pour rester performantes. Conseil : Intégrer des primitives de concurrence et de parallélisme, comme les coroutines ou des modèles d'acteurs, pour permettre aux entités d'agir simultanément dans un environnement virtuel sans ralentir les performances. Exemple : Offrir des structures de base pour le traitement parallèle des entités (parallelFor, asyncSimulate).

### 2.3.5 Typing

5. Typage dynamique avec options de typage statique

Pourquoi ? La flexibilité est cruciale dans un langage de simulation, mais des options de typage statique peuvent aider à éviter des erreurs dans des projets complexes. Conseil : Un système de typage principalement dynamique (comme Python) pour faciliter l'écriture rapide de prototypes, mais offrant la possibilité d'utiliser des types statiques pour des performances optimisées et une meilleure détection des erreurs. Exemple : Un système de typage optionnel (inspiré de TypeScript), où le typage statique peut être ajouté pour les parties critiques de la simulation.

### 2.3.6 Garbage Managenent

6. Garbage Collection pour une gestion efficace de la mémoire

Pourquoi ? Les environnements virtuels peuvent impliquer un grand nombre d'objets en mémoire, notamment lors de simulations complexes. Une gestion automatique de la mémoire évite de nombreux bugs liés aux fuites de mémoire. Conseil : Implémenter un ramasse-miettes (garbage collector) efficace pour éviter les fuites de mémoire, tout en offrant des options pour gérer manuellement certains aspects de la mémoire dans des cas de haute performance. Exemple : Laisser la gestion de la mémoire automatique par défaut, mais fournir des mécanismes pour gérer manuellement des objets critiques si nécessaire (comme allocate() et free() pour les objets spécifiques).

7. Interopérabilité avec d'autres outils et moteurs

Pourquoi ? Les créateurs d'environnements virtuels utiliseront souvent d'autres outils comme Unity, Unreal Engine ou Blender pour modéliser ou gérer des éléments de leur monde. Conseil : Offrir une interopérabilité facile avec d'autres moteurs 3D, outils de modélisation ou bibliothèques externes (comme des moteurs de rendu). Cela peut se faire via des bindings vers des langages comme C++ ou via des API externes. Exemple : Des bibliothèques intégrées pour l'importation et l'exportation de modèles 3D dans des formats standards (OBJ, FBX, etc.), et la possibilité d'intégrer MetaScript dans des moteurs comme Unity via des plugins.

### 2.3.7 Modularity

8. Modularité et réutilisabilité des composants

Pourquoi ? Les développeurs devraient pouvoir créer des modules réutilisables pour leurs simulations, que ce soit des comportements d'agents, des systèmes physiques ou des modèles 3D. Conseil : MetaScript doit encourager la création de modules réutilisables avec un système de packages et de bibliothèques, permettant aux utilisateurs de partager et d'utiliser facilement des composants développés par d'autres. Exemple : Un système de gestion de packages intégré (comme pip en Python ou npm en JavaScript) pour partager des bibliothèques de comportements d'agents, d'interactions physiques ou d'assets 3D.

### 2.3.8 Portability

### 2.3.9 Abstraction

### 2.3.10 AI Integration

Pourquoi ? L'intelligence artificielle est souvent un élément clé dans la création d'environnements virtuels interactifs. Qu'il s'agisse de simuler des personnages non-joueurs ou des agents autonomes, MetaScript doit offrir un support natif pour la création d'IA. Conseil : Intégrer des primitives pour la gestion d'agents autonomes, la prise de décision basée sur des modèles d'apprentissage automatique ou des systèmes de règles, et les interactions intelligentes avec l'environnement virtuel. Exemple : Des fonctions intégrées comme trainAI(), pathfinding(), ou behaviorTree() pour gérer l'intelligence artificielle des entités.

## 2.4 Graphical Scene Tools

## 2.5 Comparison with Programming Languages

# Chapter 3

# MetaScript Algebra

## 3.1 Basic Fundamentals

Some Basics Mahs Properties we have to know before understanfing the MetaScript Algebra

- Linear Algebra
- Diffential Equations
- Numerical Methods
- Geometry and Trigonometry
- Graph Theory
- Lagrangian and Hamiltonian Mechanics
- Fourier Analysis

## 3.2 Rules of MetaScript Algebra

Chapter 4

# Formal Grammar of the Language

# Chapter 5

# MetaCore : Graphical Integrator

## 5.1 Graphical Selector

### 5.1.1 Meta - CLI

### 5.1.2 Meta - WebView

### 5.1.3 Meta - MobileView

### 5.1.4 Meta - WindowView

### 5.1.5 Meta - TvViewer

## 5.2 Graphical Enginer

### 5.2.1 The Default Graphical Engine

### 5.2.2 Meta - for Unity

### 5.2.3 Meta - for Unreal Engine

### 5.2.4 Meta - for Godot

### 5.2.5 Meta - for Three.Js

### 5.2.6 Meta - for Phaser

### 5.2.7 Switching Graohical Engine with MetaScript

## 5.3 Graphical Templates

# Chapter 6

# The New Paradigm : Visual Oriented Programming

## 6.1 Actual Code Designs

## 6.2 Design your COde for MetaScripting

Chapter 7

# Memory Management

Chapter 8

# MSCP : MetaScript Compiler

Chapter 9

# Hardware Management with MetaScript

Chapter 10

# Vision and Scope of Language

# Bibliography