

렌더 스테이트 테이블에서 착안한 프레임-스테이트 실행 모델 기반 비주얼 노벨 저작 프레임워크

유현수, 고정국
동명대학교 컴퓨터공학과
e-mail : lioie6478@gmail.com

A Visual-Novel Authoring Framework with a Frame-State Execution Model Inspired by Render State Tables

Hyun-su Yu, Jeong-guk Koh
Dept. of Computer Engineering, Tongmyong University

요 약

비주얼 노벨(Visual Novel)은 대사에 배경·캐릭터·음향·연출을 결합한 대규모 시나리오를 저작하는데, 연출을 게임 로직에 직접 기술하면 콘텐츠가 특정 구현에 강하게 결합된다. 본 논문은 이를 해소하는 저작 프레임워크 Visual Wave(VW)를 제안·구현한다. VW는 엔진 독립 코어 라이브러리, 엔진별 런타임, VS Code 편집 지원(LSP, Language Server Protocol)[3]의 세 컴포넌트로 시나리오 DSL(Domain-Specific Language)[8]의 작성·편집·구동을 포괄하고, 콘텐츠를 시나리오·자원·시스템 세 계층으로 분리해 작가와 프로그래머의 관심사를 나눈다. 실행은 렌더 스테이트 테이블(render state table)[9]에서 착안한 프레임-스테이트 모델로, 각 채널의 상태를 선언하면 대사·선택지에서 누적 상태를 채널별 핸들러로 일괄 반영(commit)하고 상충을 최종 반영(last-write-wins)으로 해소한다. 상용 수준 비주얼 노벨[1]에 적용해 실용성을 보였으며, 변화분만 선언하는 표현은 대사 1자당 저장 용량을 이전 노드그래프[2] 방식의 약 6분의 1로 압축한다.

1. 서론

비주얼 노벨은 대사를 중심으로 배경·캐릭터·음향·연출과 플레이어 선택에 따른 분기를 결합한 장르로, 한 편은 수천 줄의 대화와 수많은 연출 지시, 다국어 텍스트, 루트 간 이동으로 이루어진다. 전통적 저작 방식은 연출 로직을 엔진 코드에 직접 작성하거나 작품마다 임시 스크립트로 기술하는 것인데, 이는 서사 콘텐츠를 특정 엔진·구현에 강하게 결합시켜 작가가 프로그래밍 세부를 알아야 하고 연출 코드의 재사용을 어렵게 한다.

본 연구는 같은 작품을 이전에 노드그래프 기반 도구로 저작한 경험에서 출발했다. 그 도구는 각 장면을 슬라이드형 UI로 편집하며 한 화면(프레임)을 전체 상태 객체로 저장해 상태가 바뀌지 않아도 프레임마다 반복 기록했고, 리소스를 Unity 에디터 직렬화 참조로 연결해야 했으며, 기계 생성 형식이라 전용 도구 안에서만 다룰 수 있었다[2]. 본 프레임워크는 대본과 리소스를 모두 하나의 텍스트 전용 언어로 표현하여, ① 변화분만 선언해 용량을 줄

이고, ② 리소스를 텍스트 경로로 기술·런타임 해석해 재바인딩·수정에 에디터 재연결이나 재빌드가 필요 없으며, ③ 평문이므로 전용 VS Code 확장으로 구문 강조·자동완성·인레이 힌트를 제공한다.

이 접근의 착안점은 실시간 그래픽스의 렌더 스테이트 테이블이다. 개발자가 블렌딩·깊이 같은 상태(state)를 설정한 뒤 드로우 콜(draw call)을 발행하면 파이프라인이 한 프레임을 그리듯[9], 비주얼 노벨의 한 화면도 여러 채널 상태가 겹쳐진 결과로 보아 각 채널 상태를 선언하고 대사·선택지에서 확정하면 화면을 선언적으로 표현할 수 있다. 본 논문의 기여는 두 가지다. 첫째, 작성·편집·구동을 포괄하고 콘텐츠를 시나리오·자원·시스템 세 계층으로 분리해 작가와 프로그래머의 관심사를 나누는 완결된 저작 프레임워크를 제안한다. 둘째, 그 실행 근간으로 비종단 명령의 상태 누적과 종단 명령에서의 일괄 반영(commit), 상충의 최종 반영(last-write-wins)으로 정의되는 프레임-스테이트 실행 모델을 제시한다.

2. 연구 방법

제안하는 Visual Wave는 시나리오의 작성부터 구동까지를 포괄하는 세 컴포넌트의 완결된 프레임워크이다(그림 1). 핵심은 특정 엔진에 의존하지 않는 코어 라이브러리로, DSL 해석과 프레임-스테이트 실행을 담당한다. 화면을 그리는 런타임 라이브러리는 이 코어를 사용하는 엔진별 바인딩으로, 본 논문은 Unity 런타임을 구현해 채널별 핸들러로 연출을 확장하였다. 코어가 엔진 독립적이라 DSL·실행 모델·자원 정의를 그대로 둔 채 런타임만 교체하면 다른 엔진에서도 동일하게 구동할 수 있으며, 이는 3절에서 Godot로 실증한다. 셋째 컴포넌트는 VS Code에서 자동완성·인레이 힌트·구문 강조를 제공하는 LSP 편집 지원 확장이다. 이처럼 세 컴포넌트가 구동·화면 구성·저작 편집을 담당하여 특정 엔진에 묶이지 않고 저작 수명주기 전체를 엔진 독립적으로 포괄한다. 저작자는 세 종류의 평문 파일을 작성하는데, 시나리오 스크립트(.vwsor)는 장면과 명령을, 자원 정의(.vwsor)는 숫자 식별자와 실제 자원 경로의 대응을, 시스템 정의(.vwsmr)는 전역 변수와 지원 언어를 담는다.

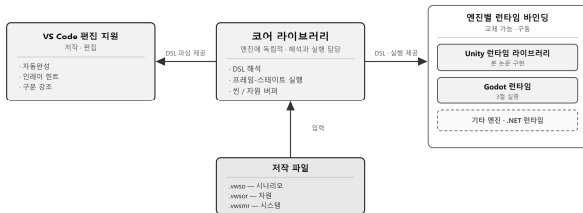


그림 1. Visual Wave 프레임워크 구성

시나리오 DSL은 한 줄을 하나의 명령으로 보고 줄 앞머리의 접두사로 종류를 판별한다. 명령은 장면 시작·종료(SS#/ES), 배경, 캐릭터 추가·삭제, 대사, 대사 배경, 선택지, 장면 점프, 배경음악·효과음, 그리고 프롭·CG·카메라·후처리 등의 연출로 나뉜다. 각 명령에는 ?로 시작하는 연출 옵션과 !로 시작하는 매크로를 덧붙일 수 있으며, 대사 명령은 !LA_ 언어 키워드마다 다른 문자열을 담아 다국어 표현을 한 줄에 표현하고, 선택지 명령은 발생 조건·선택지 대사·전역 변수 변화·점프를 한 명령에 담는다. 명령별 문법은 표 1과 같다.

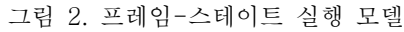
표 1. Visual Wave 시나리오 명령 분류

분류	대표 문법	의미
장면	SS#Route_NN / ES	루트·번호로 장면 시작 / 장면 종료
배경	BG_NN ?EF_Fade	배경 자원 NN을 지정 효과로 전환
캐릭터	CH_NN_EE ?RP_Speaker /	표정 EE의 캐릭터 추가·주목 / 삭제

	DCH_NN	
대사	Line_NN ?EF_... !LA_KO "..." !LA_EN "..."	화자 NN의 다국어 대사 출력
대사 배경	LineBG_NN ?EF_Fade	대사창 배경 지정
선택지	BCH ?CD_[...] !LA_... "..." !GV_... ?JMP_Route_NN	조건·대사·변수변화·점프를 담은 선택지
장면 점프	JMP_Route_NN	다른 장면·루트로 이동
음향	BGM_NN / BGM_OFF / SFX_NN	지속 음악 / 중단 / 일시 효과음
연출	PPEF_..., CG_NN, CameraEF_..., PostProcess_NN	프롭·CG·카메라·후처리 등 화면 연출
시스템	!GV_Name_Set "값"	전역 변수 조작

시나리오 스크립트는 자원을 이름이 아니라 숫자 식별자로 참조하며, 예컨대 배경은 BG_01, 캐릭터는 CH_04_01처럼 쓰고 실제 자원은 자원 정의 파일에 분리해 둔다. 자원 정의 파일은 각 식별자의 상대 경로·별칭과 캐릭터의 크기·중심점·배치 프리셋을 기록하고, 시스템 정의 파일은 전역 변수와 지원 언어를 선언한다. 이처럼 서사·연출의 시나리오 계층, 자원 바인딩의 자원 계층, 변수·언어의 시스템 계층을 분리함으로써, 자원 정의만 교체해 리소스를 바꾸거나 수천 줄의 대사가 소수의 식별자만 참조하도록 하여 재사용성을 높인다.

핵심은 한 번의 화면 갱신을 처리하는 프레임-스테이트 실행 모델이다(그림 2). 실행기는 시나리오를 순서대로 읽되 배경·캐릭터·음향·연출처럼 화면을 확정하지 않는 비종단 명령은 이번 화면 상태로 누적하고, 대사·선택지·장면 끝 같은 종단 명령에서 누적된 명령을 종류별 채널 핸들러에 순서대로 반영해 한 화면을 완성한다. 이는 여러 상태를 설정한 뒤 드로우 콜에서 한 프레임을 확정하는 렌더 스테이트 테이블의 원리와 같으며, 같은 채널의 상충 명령은 별도 제거 절차 없이 순서 기반 최종 반영으로 자연스럽게 해소된다. 각 채널은 명령 실행과 상태 초기화만 규정하는 핸들러 인터페이스의 구현이 담당하고 그 구체 구현은 게임이 제공하므로, DSL과 실행 모델을 고정한 채 연출의 실제 동작만 작품·엔진에 맞게 확장할 수 있다. 카메라 셰이크·페이드 같은 시간 경과형 연출은 상태 확정과 분리되어, 확정된 상태를 즉시 교체할지 일정 시간에 걸쳐 전개할지는 핸들러에 위임된다. 즉, last-write-wins는 각 채널의 확정 목표 상태에 적용되고 그 시간적 전개는 핸들러의 책임이다.



3. 연구 결과

표 2. 샘플 작품[1] 적용 규모

항목	값
루트 수	4(프롤로그 1, 캐릭터 루트 3)
장면 수 / 시나리오 파일 수	55 / 55
전체 명령행 (대사 명령)	3,035 (2,497)
배경 / 캐릭터 추가 / 캐릭터 삭제	38 / 79 / 39
배경음악 / 효과음 / CG	75 / 44 / 13
선택지 / 장면 점프	28 / 26
자원: 배경 / 캐릭터(이미지 바인딩)	9 / 12 (45)
자원: BGM / SFX / CG / 프리셋	10 / 13 / 18 / 13
전역 변수 / 언어	6 / 1(KO, 다국어 지원)

고 모든 자원 참조가 하나의 자원 정의 파일로 수렴하여, 자원 정의만 교체하면 시나리오를 그대로 두고 리소스를 바꿀 수 있었다. 프레임-스테이트 모델은 명령형 연출 코드 없이 배경·캐릭터·음향이 겹쳐진 화면을 구성하고, 상층 지시나 배경 전환 시 CG 초기화 같은 채널 간 정리를 예외 처리 없이 순서 기반 반영으로 해소하였다. 또한 동일한 코어·Unity 런타임·핸들러 집합을 네 루트 전체에서 수정 없이 재사용하여, 연출을 게임별 핸들러로 한 번 구현하면 작품 전체를 저작할 수 있음을 확인하였다.

한편 제안 방식의 효과를 정량적으로 확인하기 위해 동일한 작품을 이전에 저작한 노드그래프 기반 도구[2]와 객관적으로 측정 가능한 지표로 비교하였다. 이전 도구는 한 화면(프레임)을 전체 상태 객체로 저장하여 상태가 바뀌지 않아도 모든 프레임에 11개 상태 필드를 반복 기재하는 반면, 제안 방식은 변화분만 명령으로 선언한다. 두 저작물은 규모가 달라 이전 도구가 127장면·2,734프레임, 제안 방식이 55장면이므로, 규모에 영향받지 않는 정규화·비율 지표로 측정하였으며 결과는 표 3과 같다.

표 3. 동일 작품을 두 도구로 저작한 정량 비교

지표	이전 노드그래프 도구[2]	제안 방식(VW)
대사 1자당 저장 용량	약 30.0 B	약 4.7 B
상태 중복률	85.4 %	0 % (변화분만 기재)
화면 1회 기재량	11	평균 1.14

한편 동일한 한 장면을 널리 쓰이는 텍스트 기반 도구의 문법으로도 기입해 비교하였다. Ren'Py[4], Naninovel[5], Ink[6]과 제안 방식의 대사 1자당 저장 용량은 3.9~5.8 B로 비슷하고 넷 모두 노드그래프 방식보다 약 6배 압축적이어서, 저장 밀도는 텍스트 DSL 간 변

별점이 아니었다. 차별점은 설계 구조에 있다(표 4). 기존 도구가 특정 엔진에 묶이거나 Ink처럼 배경·캐릭터 연출을 전용 명령으로 다루지 않고 자원·다국어를 본문·별도 파일로 두는 반면, 제안 방식은 엔진 독립 코어에서 연출을 전용 명령으로 표현하고 자원을 숫자 식별자로 분리하며 다국어를 한 대사 줄에 함께 담는다.

표 4. 기존 저작 도구와의 구조 비교

항목	VW	Ren'Py	Naninovel	Ink	Fungus
엔진에 독립적인 코어	예 (Unity·Godot 실증)	아니오	아니오	예	아니오
배경·캐릭터 연출 전용 명령 제공	예	예	예	아니오 (태그 방식)	예(GUI)
평문 텍스트 DSL(LSP)	예	예	예	예	아니오 (GUI)
자원 참조 본문 인라인	아니오 (숫자ID 분리)	예	부분	예	—
다국어 대사 저장 파일 수	1	2	2	2	—

끝으로, 코어의 엔진 독립성을 실증하기 위해 Unity와 전혀 다른 엔진인 Godot 4.7의 Vulkan 렌더러에서 동일한 코어 라이브러리와 시나리오·자원·시스템 파일(.vwso/.vwsor/.vwsmr)을 수정 없이 재사용하는 최소 구동 사례를 구현하였다. 코어는 Unity 전용 API에 의존하지 않아 .NET 런타임으로 재컴파일되어 Godot의 C# 런타임에서 그대로 시나리오를 로드·파싱하였고, 표현 계층만 Godot 노드(배경 TextureRect, 대사 Label)로 새로 구현해 프레임-스테이트 실행 모델의 상태 누적·확정 규칙을 재현하였다. 그 결과 동일한 대본이 Unity가 아닌 엔진에서 배경·대사로 렌더링됨을 확인하였다(그림 3). 이는 DSL·실행 모델·자원 정의를 고정된 채 표현 계층만 교체하면 서로 다른 엔진에서 같은 시나리오를 구동할 수 있음을 보인다. 다만 본 실증은 배경·대사 채널에 한정된 최소 사례로, 캐릭터·음향·연출 등 나머지 채널은 엔진별 자원 형식에 맞춘 추가 구현을 필요로 한다.

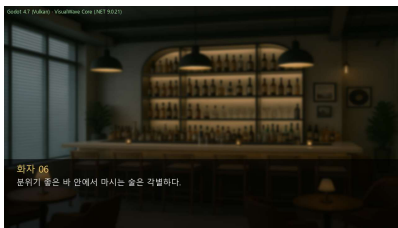


그림 3. 엔진 독립성 실증(Godot 엔진 버전)

4. 결론

본 논문은 비주얼 노벨 시나리오의 작성·편집·구동을 포괄하는 완결된 저작 프레임워크와 그 실행 근간인 프레임-스테이트 실행 모델을 제안·구현하였다. 프레임워크는 엔진 독립 코어, 각 엔진 종속 런타임, VS Code 편집 지원의 세 컴포넌트로 이루어지며, 콘텐츠를 시나리오·자원·시스템 세 계층으로 나누어 작가·프로그래머의 관심사를 분리한다. 실행 모델은 렌더 스테이트 테이블에서 착안하여 비종단 명령을 프레임 상태로 누적하다가 대사·선택지에서 일괄 반영하고, 상층을 순서 기반 최종 반영으로 해소한다. 이를 상용 수준 비주얼 노벨에 적용하여 분기형 서사를 하나의 자원 정의와 재사용 가능한 런타임으로 저작할 수 있음을 확인하였고, 대사 1자당 저장 용량을 이전 노드그래프 방식의 약 6분의 1로 압축하며 코어 측 실행 오버헤드가 화면 1회당 약 0.16 ms에 그침을 보였다. 또한 동일한 코어·DSL·자원을 Godot에서 수정 없이 구동해 코어의 엔진 독립성을 실증하였다. 향후 과제로는 시나리오·자원 정합성 사전 검사 컴파일 단계 도입, 문자열 별칭 식별자 병기, 캐릭터·연출 등 전 채널을 포함한 다중 엔진 지원, 사용자 평가(User Study)를 통한 저작 생산성의 정량적 검증을 들 수 있다.

참고문헌

- [1] ALCOHOL FREE, 비주얼 노벨 게임.
- [2] 유현수, "AlcoholFree-PlotTree," GitHub, 2026. <https://github.com/HyunsuYu/AlcoholFree-PlotTree>
- [3] Microsoft, "Language Server Protocol Specification," <https://microsoft.github.io/language-server-protocol/>
- [4] T. Rothamel, "Ren'Py Visual Novel Engine," <https://www.renpy.org/>
- [5] Naninovel, "Naninovel — Visual Novel Engine for Unity," <https://naninovel.com/>
- [6] inkle, "ink — inkle's narrative scripting language," <https://www.inklestudios.com/ink/>
- [7] snozbot, "Fungus — Visual storytelling and scripting tool for Unity," <https://github.com/snozbot/fungus/>
- [8] M. Fowler, Domain-Specific Languages. Addison-Wesley, 2010.
- [9] T. Akenine-Möller, E. Haines, N. Hoffman, et al., Real-Time Rendering, 4th ed. CRC Press, 2018.