**개발문서**

1. **소개**
2. OS: Window10
3. 엔진: 유니티
4. 버전: 2021.1.17
5. 개발 기간: 2021.09.14 ~ 2021.12.16
6. **캐릭터 설명**
7. 구현 요구 사항
8. 움직임
9. 스킬
10. 캐릭터간 상호작용
11. **에이전트 설계 구조**

* 에이전트는 **컴포넌트 상호간 직접 참조를 최대한 줄여 설계**.
* 낮은 커플링 정도를 유지, **객체간 상호작용을 한 클래스에서 관리.**
* **시스템 설계 스케치**로 캐릭터와 외부 객체의 소통 방식을 거시적으로 파악. **클래스 다이어그램**을 통해 구체화. 마지막으로 **커뮤니케이션 다이어그램**으로 클래스간 메시지 소통 과정을 파악.

1. **에이전트 아키텍처 프로토타입**
2. **에이전트 아키텍처의 프로토타입은 환경을 포함하여 4가지를 고려하여 구성**
   1. 판단: 의사결정을 담당하는 객체
   2. 내부 상호작용: 지식을 생성하여 의사 결정에 영향을 미치는 블랙보드 기능 및 운동 생성을 하여 게임 세계와 소통하는 이펙터 역할.
   3. 외부 상호작용: 환경이 에이전트와 소통하기 위한 통로 역할

|  |
| --- |
|  |
| 그림1. 에이전트 아키텍처와 정보의 흐름 |

1. **클래스 다이어그램**
   1. **플레이어**
      1. 유저가 직접 플레이어를 조작하기 때문에 플레이어의 **판단** 부분은 InputManager가 대체.
      2. 블랙보드와 이펙터 역할을 하는 **내부 상호작용** 부분은 PlayerController, SkillController, MovementController, Animator, StateBehavior, EffectController로 구성. 내부 상호작용 내 클래스간 상호작용은 PlayerController를 중개자 패턴으로 생각하여 설계. 단, Animator는 상태머신 역할을 동시에 하기 때문에 동일한 수준의 접근 권한을 얻음.
      3. 환경이 에이전트와 소통을 하기위한 통로인 **외부 상호작용** 부분은 플레이어의 스탯 상태를 저장하는 AbilityStatus와 버프 & 디버프 관련 클래스인 StatusAilment, Buffable로 구성.

|  |
| --- |
|  |
| 그림2. 플레이어 클래스 다이어그램 |

* + 1. MovementController

- 플레이어와 몬스터 둘다 판단 모듈에서 움직임을 제어하기 때문에 하나의 클래스로 동작시킬 수 있다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 3 움직임 컨트롤러 클래스 다이어그램

* + 1. AgentController

- 에이전트 컨트롤러에서 인터페이스를 정의. 서브 클래스에서 논리 구조를 구현.

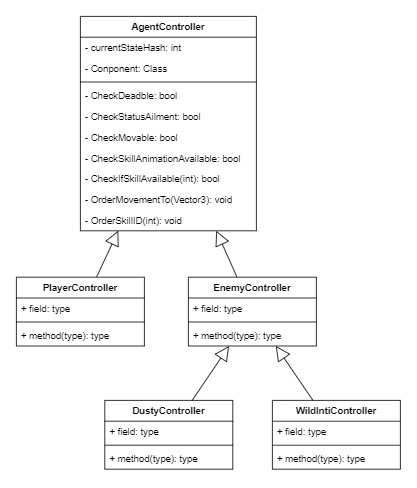


그림 4 에이전트 컨트롤러 클래스 다이어그램

1. 커뮤니케이션 다이어그램
2. 캐릭터 상태 구조
   1. 애니메이션 상태, 애니메이션 상태머신

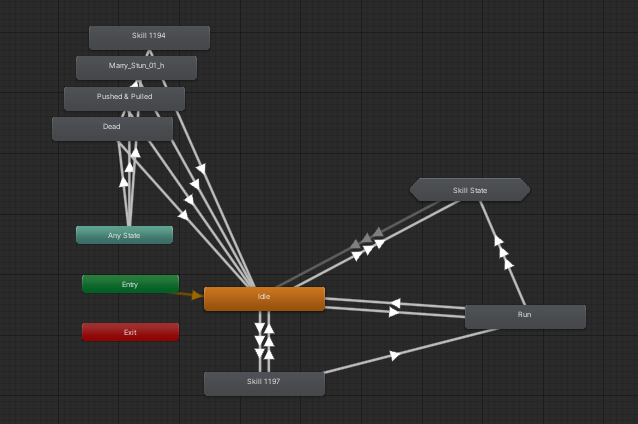


그림 5 에이전트 애니메이터

* + 1. 사용 이유

- 기존 유니티의 애니메이터 컴포넌트는 상태머신 역할을 할 수 있으며, State Machine Behavior 기능을 제공. 따라서 엔진의 기능을 활용하기로 결정.

* + 1. 장점

- 애니메이션과 상태를 기본적으로 1:1 매칭 가능. 파라미터를 추가하면 1:N 매칭이 가능하게 되어 확장성 있는 설계 가능.

* + 1. 단점

- State Machine Behavior 기능을 레이어에 넣을 수도 있지만 애니메이션 상태에 넣을 수도 있음. 이는 코드를 분산시킬 수 있어 가독성 저하를 불러 일으킴.

1. 스킬 사용 구조
   1. 클래스 다이어그램

- 스킬 컨트롤러에서 인터페이스와 부분 구현. 서브 클래스에서 State Machine Behavior와 소통하는 부분을 상세 구현.

- UploadeSkillIDToAnimatorParameter 함수는 리소스 폴더에서 스킬 데이터 스크립터블 오브젝트를 불러와서 각 객체에 맞는 스킬 아이디를 불러온다.

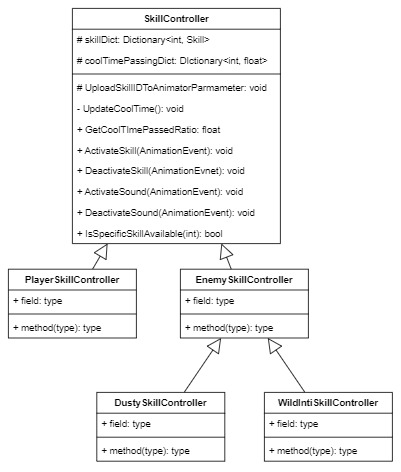


그림 6. 스킬 컨트롤러 클래스 다이어그램

* 1. 커뮤니케이션 다이어그램

1. 이팩트 사용 구조
   1. 클래스 다이어그램

- 애니메이터 내 AnimationEvent를 활용한다. AnimationEvent에서 이벤트를 Invoke하면 구체화된 EffectController 클래스의 함수들이 Listen 후 실행한다.

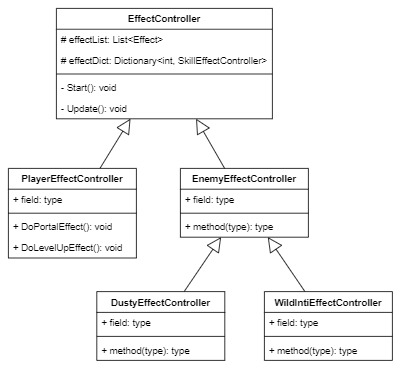


그림 7. 이펙트 컨트롤러

* 1. 커뮤니케이션 다이어그램

1. 이동기 사용 구조
   1. 클래스 다이어그램
   2. 커뮤니케이션 다이어그램
2. 타 객체와 상호작용 구조
   1. AbilityStatus
   2. Buffable
   3. StatusAilment