NEXON
DEVELOPERS
CONFERENCE

〈하스스톤〉 강화학습 환경 개발기

- 0티어 덱을 만들기 위해 떠나는 모험

넥슨 코리아
옥찬호

NDC NEXON DEVELOPERS CONFERENCE

발표자 소개

utilForever@gmail.com

utilForever

옥찬호 (Chris Ohk)

- 넥슨 코리아 게임 프로그래머
- 마이크로소프트 Developer Technologies MVP
- 페이스북 그룹 C++ Korea 대표
- NDC 2018 〈유체역학 엔진 개발기〉 발표
- 언젠가 모든 카드를 황금 카드로 만들고 싶은 하스스톤 헤비 과금러 (황금덱 다수 보유)

NEXON
DEVELOPERS
CONFERENCE

안내

이 발표는 개인 연구 프로젝트입니다.

블리자드 엔터테인먼트의 후원, 보증, 제휴 관계는 없습니다.

NEXON
DEVELOPERS
CONFERENCE

안내

이 발표는 강화학습을 하기 위한 환경을 만드는 과정을 설명합니다.

주로 하스스톤 게임 개발에 관한 내용과 코드를 다루고 있습니다.

하스스톤 게임을 알고 있지 않다면 이해하는데 어려움이 있을 수 있습니다.

NEXON
DEVELOPERS
CONFERENCE

목차

NEXON
DEVELOPERS
CONFERENCE

1. 들어가며

- 감사의 말
- 하스스톤 게임 소개
- 하스스톤을 선택한 이유

감사의 말

- 발표할 수 있도록 도움을 주신 **전이삭 님**께 감사드립니다.
- RosettaStone 프로젝트를 함께 만들어 나가는 팀원분들께 감사드립니다.
  - 김영중 님, 전승현 님, 김성현 님, 김형찬 님
    박유한 님, 최정현 님, 태인우 님, 박정호 님
    송태현 님, 이도운 님, 정연수 님, 조수하 님
- 그 외 프로젝트에 기여해주시는 모든 분들께 감사드립니다.


HEARTHSTONE
HEROES OF WARCRAFT

하스스톤

- 블리자드 엔터테인먼트가 개발한 디지털 카드 수집 게임
- 2013년 3월 페니 아케이드 엑스포에서 처음 발표
- 2014년 3월 13일 PC 버전부터 출시
- 현재까지 11개의 확장팩과 4개의 모험 모드가 추가됨
- 2018년 11월 발표 기준 전세계 이용자수 1억명 돌파

https://ko.wikipedia.org/wiki/하스스톤

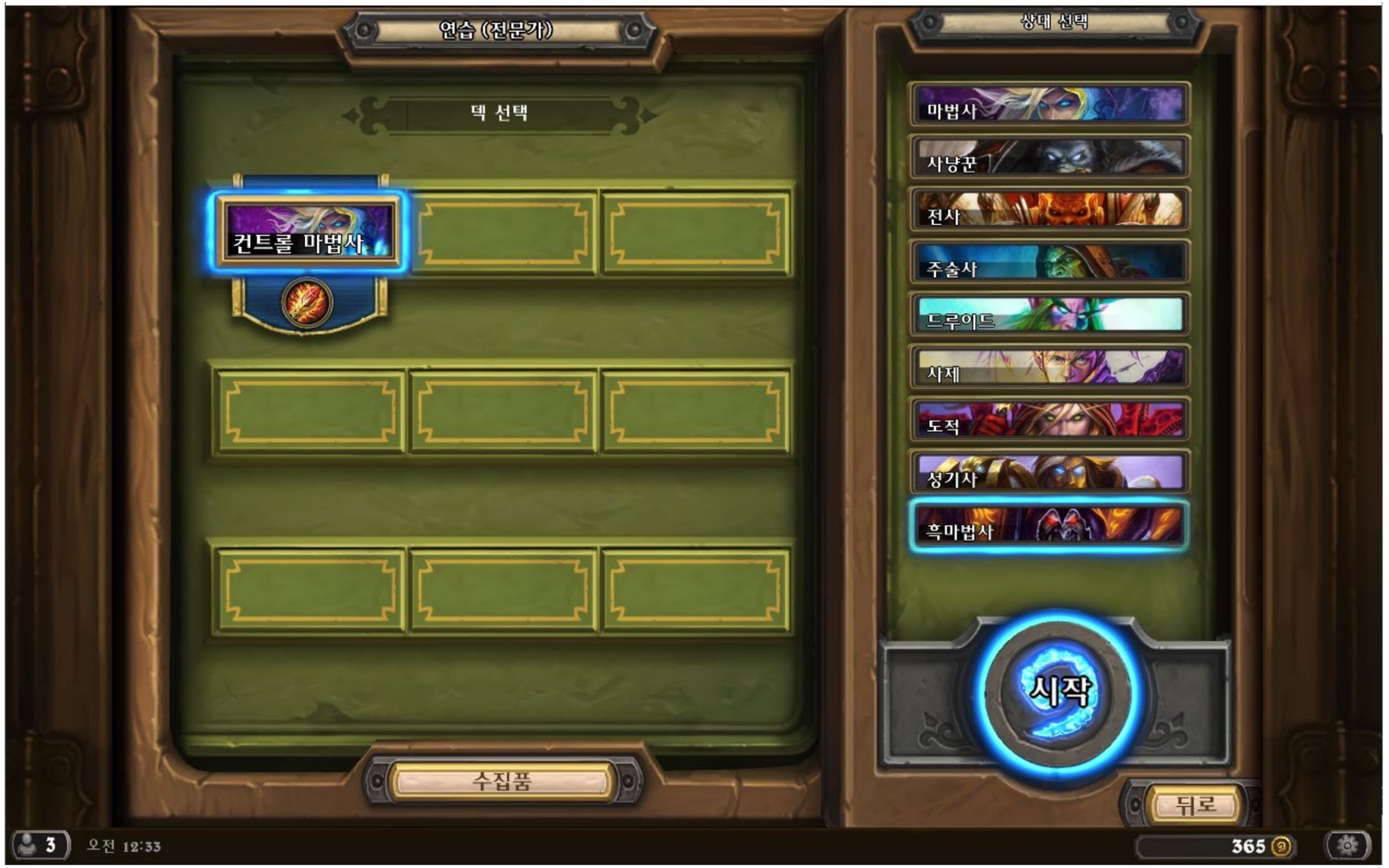

SBS뉴스
SBS뉴스
“그런데 말입니다”

Q : 왜 하스스톤을 선택하게 되었나요?

A : 아, 그게요…

저는 평소에 다양한 덱을 테스트해보곤 하는데,

연습 모드로 컴퓨터와 대결을 많이 합니다.



컴퓨터의 플레이는 대부분 괜찮습니다. 그런데…

???

??????????????

짜증이 납니다.

그러던 도중 ‘알파고’와 ‘알파스타’를 보고 취해버린 저는 하스스톤도 저렇게 할 수 있지 않을까라는 생각을 하게 됩니다.

목표

- 강화학습을 통해
  - 영리한 플레이를 할 수 있게 만들어 보자.
    (하스스톤 프로게이머와 대적할 만한 수준)
  - 승률이 높은 덱을 만들어 보자.
    (1티어 덱보다 높은 승률 : 〉= 60%)
  - 플레이어가 덱을 만들 때 어떤 카드가 없을 경우
    대체 카드를 추천하는 기능을 제공해 보자.

https://hsreplay.net/

NEXON
DEVELOPERS
CONFERENCE

2. 하스스톤 강화학습

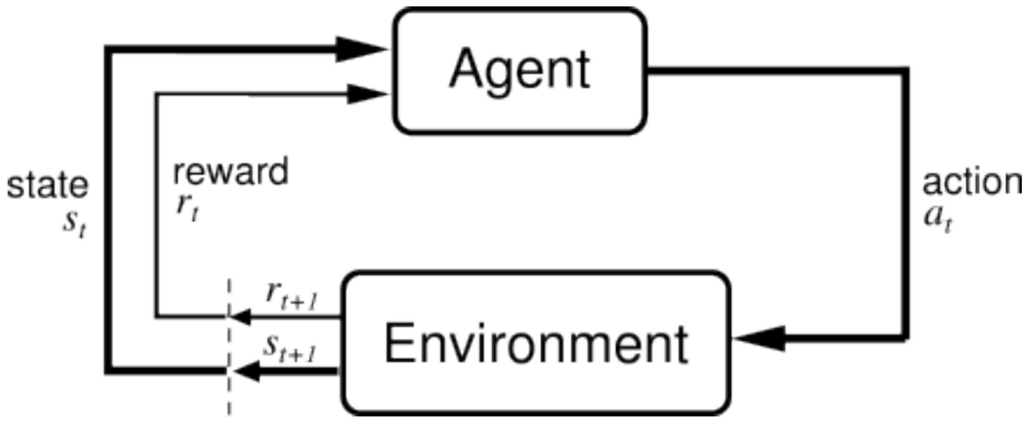
- 하스스톤이 알파고, 알파스타와 다른 점
- 하스스톤 강화학습 연구 현황

강화학습

- 아이가 첫걸음을 떼는 과정
  - 아이는 걷는 것을 배운 적이 없습니다.
  - 아이는 스스로 이것저것 시도해 보다가 우연히 걷게 됩니다.
  - 자신이 하는 행동과 걷게 된다는 보상 사이의 상관관계를 모르는 아이는 다시 넘어집니다.
  - 시간이 지남에 따라 그 관계를 학습해서 잘 걷게 됩니다.

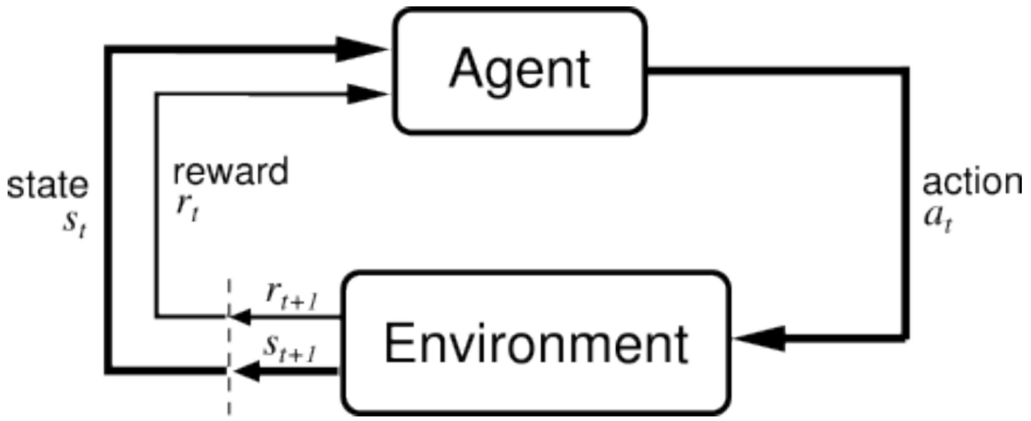
강화학습

- 에이전트는 사전 지식이 없는 상태에서 학습합니다.
- 에이전트는 자신이 놓인 환경에서 자신의 상태를 인식한 후 행동합니다.
- 환경은 에이전트에게 보상을 주고 다음 상태를 알려줍니다.
- 에이전트는 보상을 통해 어떤 행동이 좋은 행동인지 간접적으로 알게 됩니다.



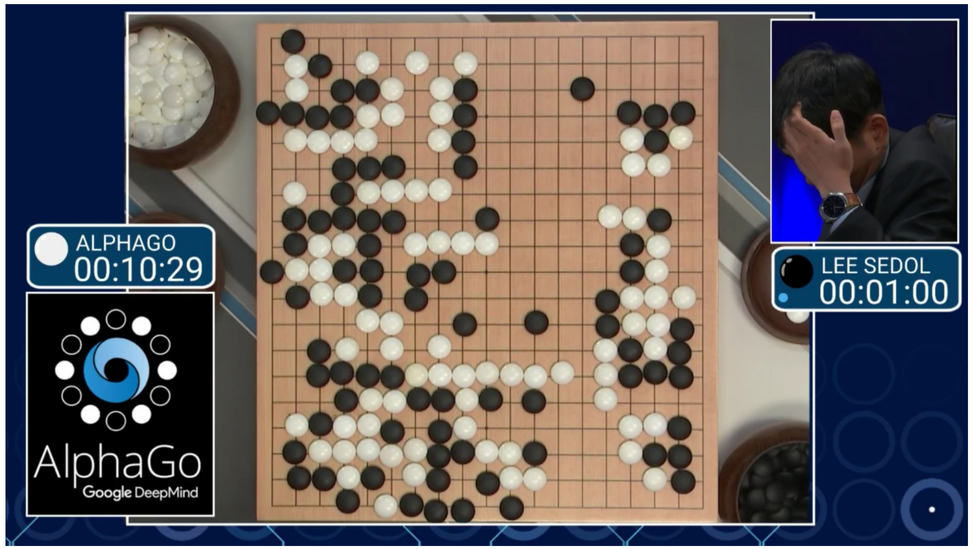
강화학습

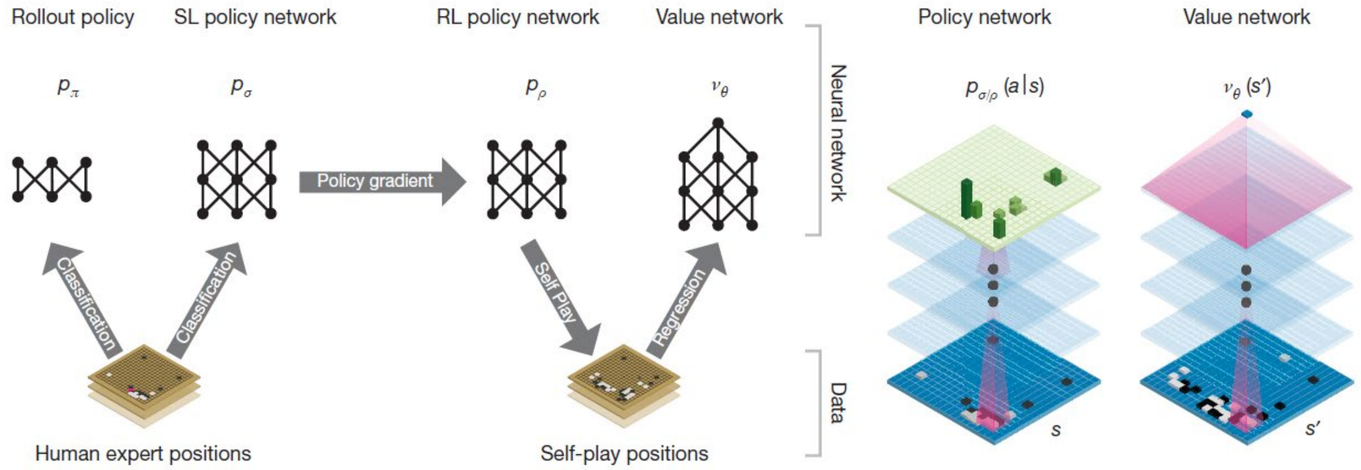
- 에이전트는 사전 지식이 없는 상태에서 학습합니다.
- 에이전트는 자신이 놓인 환경에서 자신의 상태를 인식한 후 행동합니다.
- **환경은 에이전트에게 보상을 주고 다음 상태를 알려줍니다. (오늘의 주제)**
- 에이전트는 보상을 통해 어떤 행동이 좋은 행동인지 간접적으로 알게 됩니다.



알파고

- 바둑판 위에 모든 상황에 대한 정보가 있습니다.
- 두 선수가 돌을 하나씩 번갈아 놓으며 진행합니다.





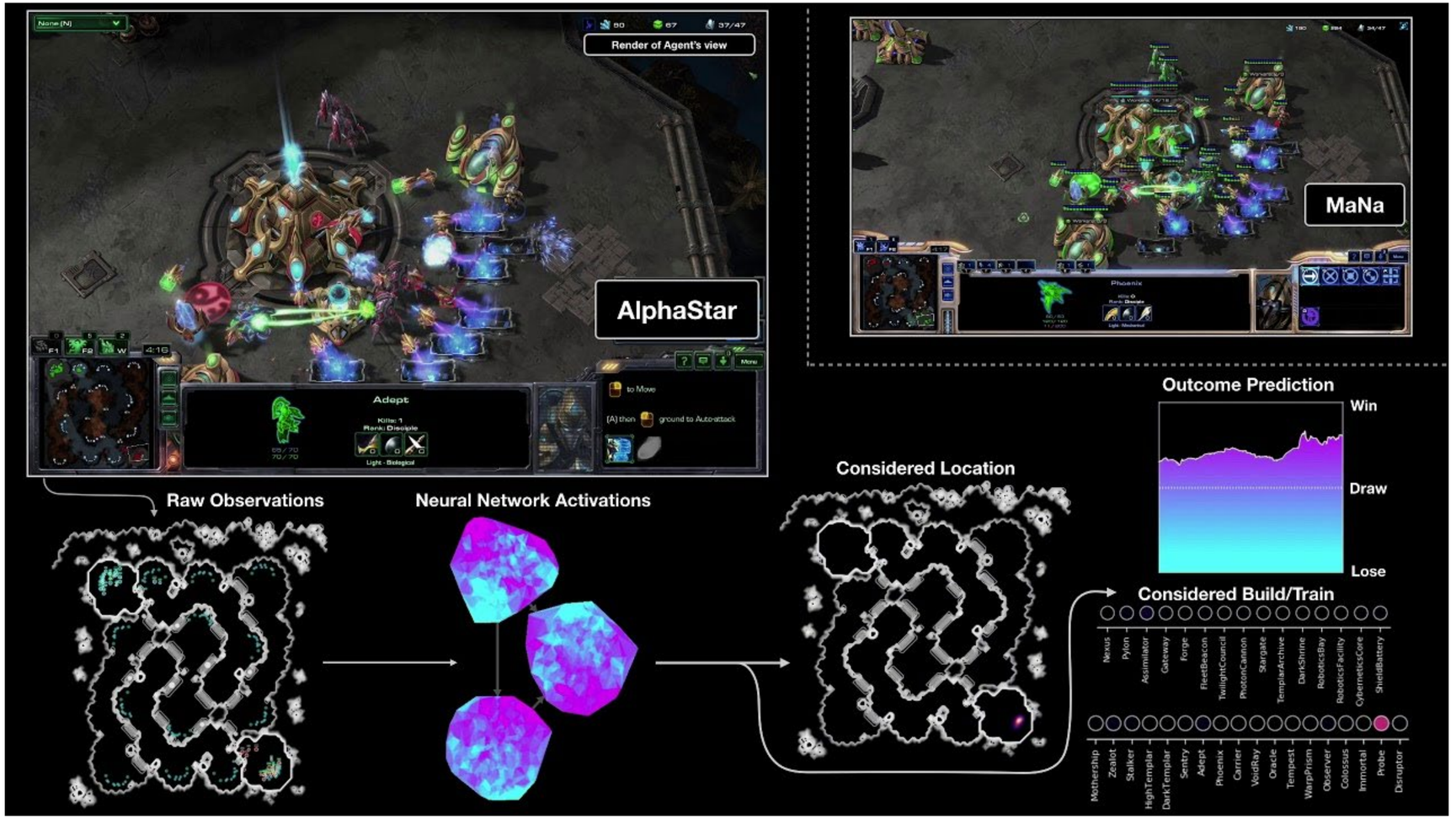
https://shuuki4.wordpress.com/2016/03/11/alphago-alphago-pipeline-헤집기/

알파스타

- 실시간 전략 게임(RTS)이며 불완전한 정보가 주어집니다.
- 유닛의 시야 범위 안에서만 정보를 얻을 수 있으며,
  정찰을 보내 탐색전을 펼치며 상대방의 상황을 추론해야 합니다.
- 실시간으로 넓은 전장에서 수백개의 유닛과 건물을 제어하며
  다양한 조합을 통해 전략을 짜야 합니다.

https://www.bloter.net/archives/329970

알파스타



하스스톤

- 카드 수집 게임(CCG)이며 불완전한 정보가 주어집니다.
  - 내 덱에서 다음에 무슨 카드가 나올지 예측할 수 없습니다.
  - 상대방이 현재 손에 무슨 카드를 들고 있는지 예측할 수 없습니다.
  - 상대방 덱에서 다음에 무슨 카드가 나올지 예측할 수 없습니다.
- 알파고나 알파스타와 달리 자신의 정보조차 불완전합니다.

하스스톤

하스스톤

- 무작위 카드가 많아서 결과를 추론하기 힘듭니다.

하스스톤

하스스톤 강화학습 연구 현황

- Świechowski, Maciej, Tomasz Tajmajer, and Andrzej Janusz. “Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms.” 2018 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2018.
- Zhang, Shuyi, and Michael Buro. "Improving hearthstone AI by learning high-level rollout policies and bucketing chance node events." 2017 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2017.
- Kachalsky, Ilya, Ilya Zakirzyanov, and Vladimir Ulyantsev. "Applying reinforcement learning and supervised learning techniques to play Hearthstone." 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2017.

NEXON
DEVELOPERS
CONFERENCE

3. 하스스톤 게임 개발

- 하스스톤을 개발하게 된 이유
- 하스스톤의 구조
- 카드 구현
- 카드 효과 구현
- 게임 구현

게임을 활용해 강화학습을 하려면

- 첫번째 방법 : 게임 회사에서 제공하는 API를 사용합니다.

게임을 활용해 강화학습을 하려면

- 첫번째 방법 : 게임 회사에서 제공하는 API를 사용합니다.
  - 스타크래프트 2는 있는데, 하스스톤은 없습니다. (실패)

게임을 활용해 강화학습을 하려면

- 첫번째 방법 : 게임 회사에서 제공하는 API를 사용합니다.
  - 스타크래프트 2는 있는데, 하스스톤은 없습니다. (실패)
- 두번째 방법 : 게임을 후킹(Hooking)해 알아낸 정보를 사용합니다.

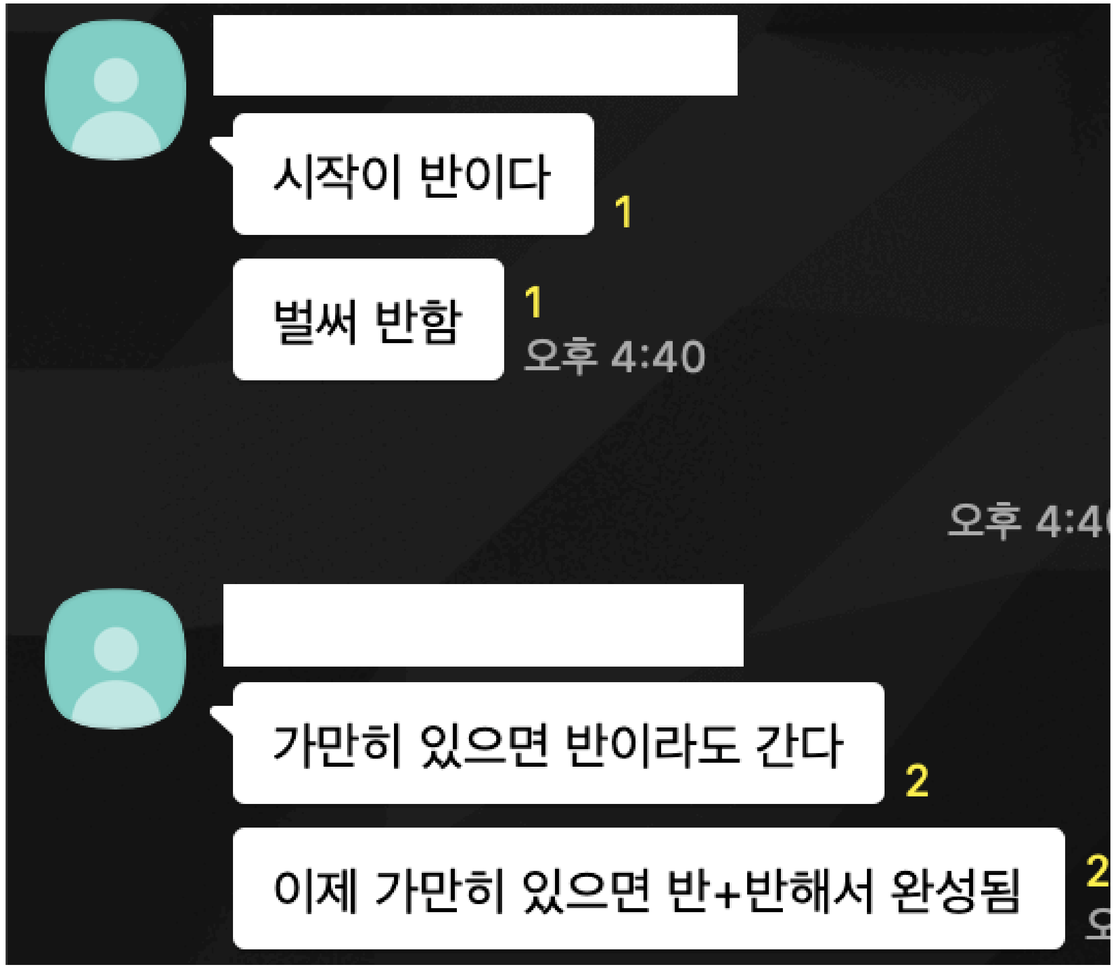
게임을 활용해 강화학습을 하려면

- 첫번째 방법 : 게임 회사에서 제공하는 API를 사용합니다.
  - 스타크래프트 2는 있는데, 하스스톤은 없습니다. (실패)
- 두번째 방법 : 게임을 후킹(Hooking)해 알아낸 정보를 사용합니다.
  - 서비스 중인 게임에 영향을 주는 행위는 불법입니다. (실패)

게임을 활용해 강화학습을 하려면

- 첫번째 방법 : 게임 회사에서 제공하는 API를 사용합니다.
  - 스타크래프트 2는 있는데, 하스스톤은 없습니다. (실패)
- 두번째 방법 : 게임을 후킹(Hooking)해 알아낸 정보를 사용합니다.
  - 서비스 중인 게임에 영향을 주는 행위는 불법입니다. (실패)
- 세번째 방법 : 하스스톤을 직접 만들어 사용합니다.
  - 아, 이 방법 만큼은 쓰고 싶지 않았는데…
  - 근데 대안이 없습니다. 어쩌겠습니까, 맨 땅에 헤딩하는 수 밖에… ㅠㅠ;

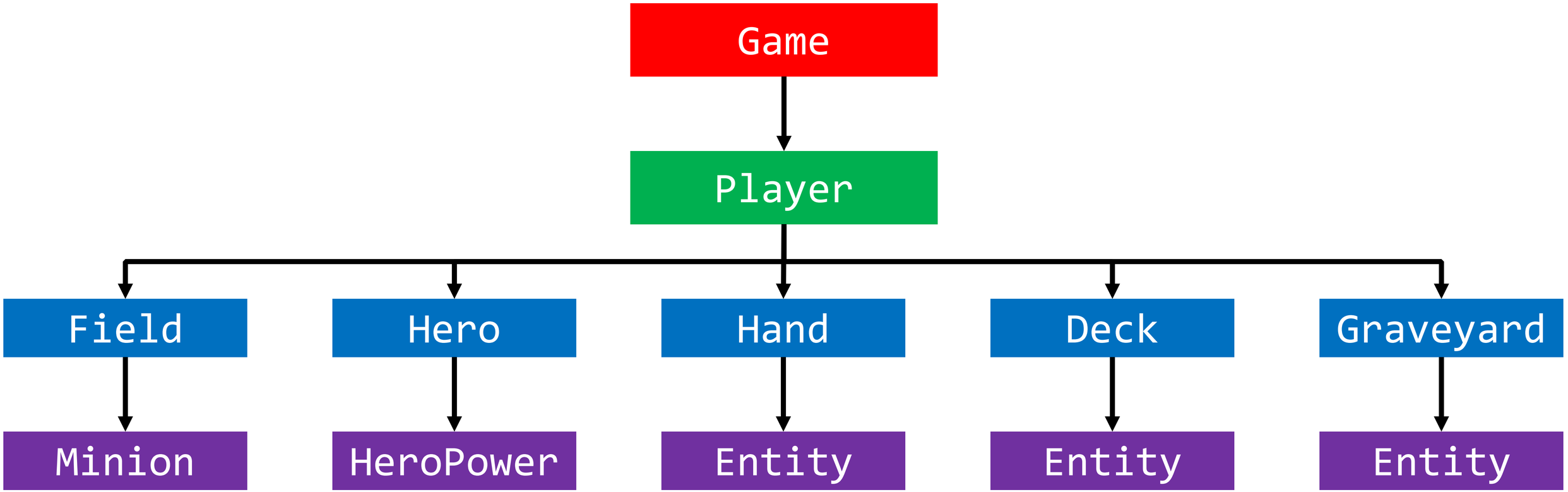
맨 땅에 헤딩해 봅시다.



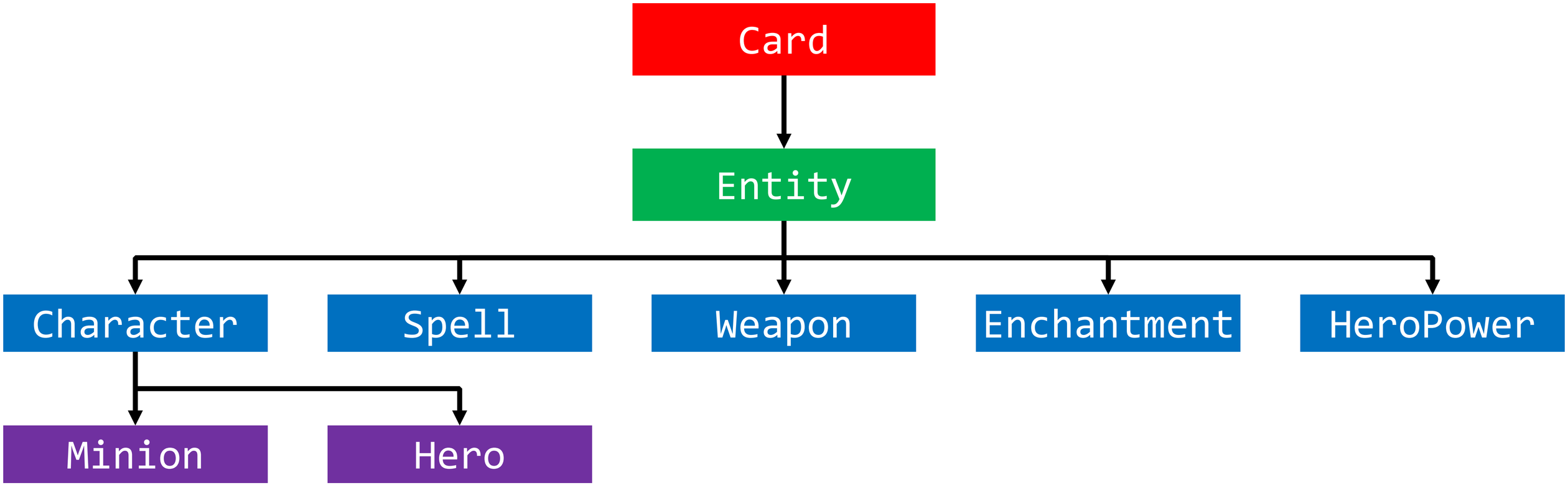
하스스톤의 구조

- 전장 (Field)
- 영웅 (Hero)
- 손 (Hand)
- 영웅 능력 (Hero Power)
- 덱 (Deck)
- 묘지 (Graveyard)

하스스톤의 구조

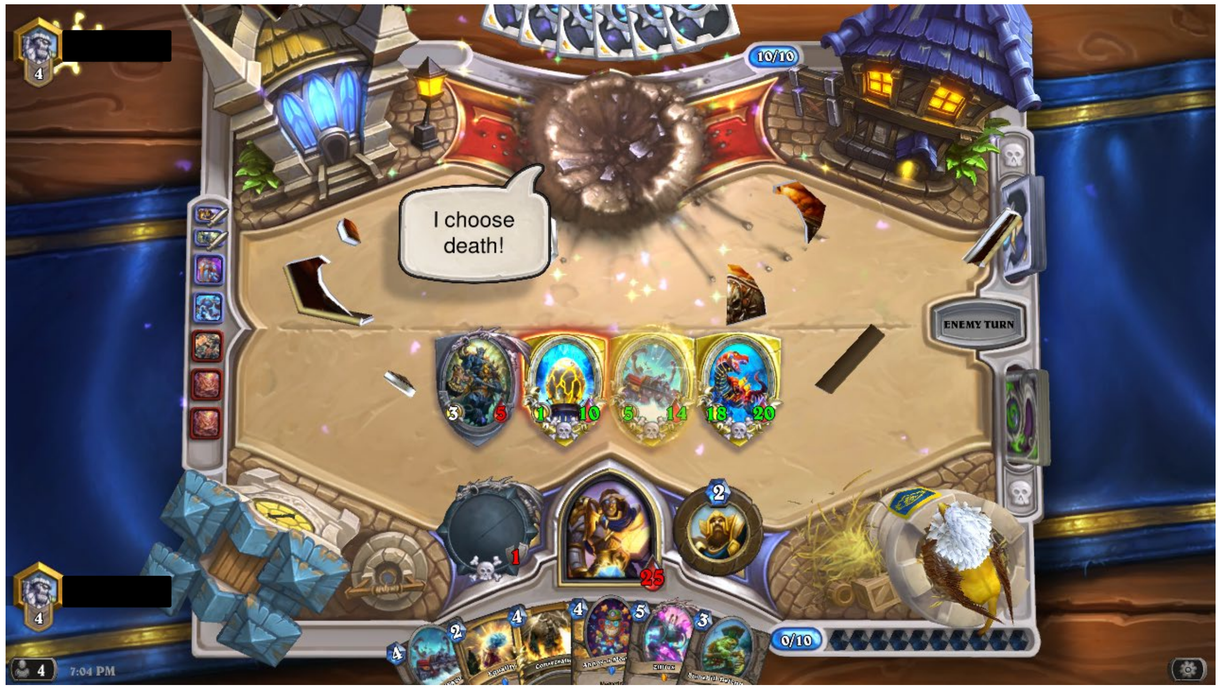


하스스톤의 구조



카드와 엔티티

- 카드 : 순수한 카드 정보만을 갖고 있는 개체
- 엔티티 : 게임에서 사용하기 위해 카드를 기반으로 새롭게 만든 개체



```
Entity* Entity::GetFromCard(Player& player, Card&& card) {
    Entity* result;
    switch (card.GetCardType()) {
        case CardType::HERO:
            result = new Hero(player, card); break;
        case CardType::HERO_POWER:
            result = new HeroPower(player, card); break;
        case CardType::MINION:
            result = new Minion(player, card); break;
        case CardType::SPELL:
            result = new Spell(player, card); break;
        case CardType::WEAPON:
            result = new Weapon(player, card); break;
    }

    // Set entity ID
    result->id = player.GetGame()->GetNextID();
    return result;
}
```

카드 구현

- 카드 게임에서 카드가 없으면 아무런 의미가 없습니다.
- 카드를 구현할 때 고려해야 할 부분
  - 카드 데이터를 어떻게 구할 것인가
  - 카드 데이터를 어떻게 가져올 것인가
  - 카드 효과를 어떻게 인식할 것인가

카드 데이터 처리

- 하스스톤에는 다양한 카드가 존재하며 데이터가 각각 다릅니다.

카드 데이터 처리

- 하스스톤에는 다양한 카드가 존재하며 데이터가 각각 다릅니다.
  - 모든 카드는 마나 코스트를 갖습니다.

카드 데이터 처리

- 하스스톤에는 다양한 카드가 존재하며 데이터가 각각 다릅니다.
  - 모든 카드는 마나 코스트를 갖습니다.
  - 하수인 카드는 공격력과 체력을 갖습니다.

카드 데이터 처리

- 하스스톤에는 다양한 카드가 존재하며 데이터가 각각 다릅니다.
  - 모든 카드는 마나 코스트를 갖습니다.
  - 하수인 카드는 공격력과 체력을 갖습니다.
  - 무기 카드는 공격력과 내구도를 갖습니다.

카드 데이터 처리

- 하스스톤에는 다양한 카드가 존재하며 데이터가 각각 다릅니다.
  - 모든 카드는 마나 코스트를 갖습니다.
  - 하수인 카드는 공격력과 체력을 갖습니다.
  - 무기 카드는 공격력과 내구도를 갖습니다.
  - 어떤 카드는 어빌리티를 갖습니다.

카드 데이터 처리

- 하스스톤에는 다양한 카드가 존재하며 데이터가 각각 다릅니다.
  - 모든 카드는 마나 코스트를 갖습니다.
  - 하수인 카드는 공격력과 체력을 갖습니다.
  - 무기 카드는 공격력과 내구도를 갖습니다.
  - 어떤 카드는 어빌리티를 갖습니다.
  - 어떤 카드는 특별한 능력을 갖습니다.

카드 데이터 처리

- 하스스톤에는 다양한 카드가 존재하며 데이터가 각각 다릅니다.
  - 모든 카드는 마나 코스트를 갖습니다.
  - 하수인 카드는 공격력과 체력을 갖습니다.
  - 무기 카드는 공격력과 내구도를 갖습니다.
  - 어떤 카드는 어빌리티를 갖습니다.
  - 어떤 카드는 특별한 능력을 갖습니다.
- 다양한 카드 데이터를 어떻게 가져올 수 있을까요?

HearthstoneJSON

- https://hearthstonejson.com/
- 하스스톤 카드 데이터를 JSON 형태로 제공하는 사이트
- 패치가 될 때마다 최신 카드 데이터를 제공합니다.
- 카드의 동작을 구현하는데 필요한 **(거의)** 모든 데이터가 있습니다.

개발이 오래 걸리는 원인
이유는 잠시 후에…

예제 : ‘리로이 젠킨스’ 카드


```
{
    "id": "EX1_116",
    "name": "Leeroy Jenkins",
    "text": "Charge. Battlecry: Summon two 1/1 Whelps for your opponent.",
    "attack": 6,
    "cardClass": "NEUTRAL",
    "collectible": true,
    "cost": 5,
    "elite": true,
    "faction": "ALLIANCE",
    "health": 2,
    "mechanics": [
        "BATTLECRY",
        "CHARGE"
    ],
    "rarity": "LEGENDARY",
    "set": "EXPERT1",
    "type": "MINION"
}
```


예제 : '리로이 젠킨스' 카드


```
{
    "id": "EX1_116",
    "name": "Leeroy Jenkins",
    "text": "Charge. Battlecry: Summon two 1/1 Whelps for your opponent.",
    "attack": 6,
    "cardClass": "NEUTRAL",
    "collectible": true,
    "cost": 5,
    "elite": true,
    "faction": "ALLIANCE",
    "health": 2,
    "mechanics": [
        "BATTLECRY",
        "CHARGE"
    ],
    "rarity": "LEGENDARY",
    "set": "EXPERT1",
    "type": "MINION"
}
```


json

- https://github.com/nlohmann/json
- 모던 C++을 위한 JSON 파싱 라이브러리
- 사용하기 쉽고 헤더 파일 하나만 추가하면 끝!

```
#include <nlohmann/json.hpp>

// for convenience
using json = nlohmann::json;
```

```
// Read card data from JSON file
std::ifstream cardFile(RESOURCES_DIR "cards.json");
nlohmann::json j;
cardFile >> j;

cards.reserve(j.size());
for (auto& cardData : j)
{
    const std::string name = cardData["name"].is_null()
                                 ? ""
                                 : cardData["name"].get<std::string>();

    const std::string text = cardData["text"].is_null()
                                 ? ""
                                 : cardData["text"].get<std::string>();
}
```

카드 데이터 처리

- 카드 타입에 따라 다양한 종류의 데이터가 문자열 형태로 저장되어 있습니다.
- 개발할 때 다음 사항들을 고려했습니다.
  - 키워드들을 열거체로 관리할 수 있으면 좋겠습니다.
    (매 번 문자열로 비교하는 것은 매우 비효율적입니다.)
  - 문자열을 열거체로 변환해서 저장할 수 있으면 좋겠습니다.
    (키워드가 많아 switch-case문으로 관리하기 힘듭니다.)
- 타입마다 어떤 종류가 있는지 어떻게 알 수 있을까요?
  그리고 열거체로 어떻게 변환해서 저장할 수 있을까요?

python-hearthstone

- https://github.com/HearthSim/python-hearthstone
- 카드 데이터에서 사용하는 다양한 열거체 타입이 정리되어 있습니다.

```
class Rarity(IntEnum):
    """TAG_RARITY"""

    INVALID = 0
    COMMON = 1
    FREE = 2
    RARE = 3
    EPIC = 4
    LEGENDARY = 5
```

better-enums

- https://github.com/aantron/better-enums
- 문자열 ↔ 열거체 변환을 지원하는 라이브러리

```
#include <iostream>
#include "enum.h"

BETTER_ENUM(Word, int, Hello, World)

int main() {
    std::cout << (+Word::Hello)._to_string() << ", "
              << (+Word::World)._to_string() << "!" << std::endl;
}
```

```
const Rarity rarity =
    cardData["rarity"].is_null()
        ? +Rarity::FREE
        : Rarity::_from_string(
          cardData["rarity"].get<std::string>().c_str());
const Faction faction =
    cardData["faction"].is_null()
        ? +Faction::NEUTRAL
        : Faction::_from_string(
          cardData["faction"].get<std::string>().c_str());
const CardSet cardSet =
    cardData["set"].is_null()
        ? +CardSet::NONE
        : CardSet::_from_string(
          cardData["set"].get<std::string>().c_str());
```

카드 효과 처리

- 하스스톤에는 고유한 효과를 갖는 카드가 많습니다.

카드 효과 처리

- 하스스톤에는 고유한 효과를 갖는 카드가 많습니다.
  - 일부 하수인들은 능력을 발동하려면 조건이 필요합니다.

카드 효과 처리

- 하스스톤에는 고유한 효과를 갖는 카드가 많습니다.
  - 일부 하수인들은 능력을 발동하려면 조건이 필요합니다.
  - 일부 하수인들은 필드에 낼 때 효과를 발동합니다.

카드 효과 처리

- 하스스톤에는 고유한 효과를 갖는 카드가 많습니다.
  - 일부 하수인들은 능력을 발동하려면 조건이 필요합니다.
  - 일부 하수인들은 필드에 낼 때 효과를 발동합니다.
  - 일부 하수인들은 죽을 때 효과를 발동합니다.

카드 효과 처리

- 하스스톤에는 고유한 효과를 갖는 카드가 많습니다.
  - 일부 하수인들은 능력을 발동하려면 조건이 필요합니다.
  - 일부 하수인들은 필드에 낼 때 효과를 발동합니다.
  - 일부 하수인들은 죽을 때 효과를 발동합니다.
  - 비밀은 특정 조건에 부합하는 상황이 일어나면 저절로 발동합니다.

카드 효과 처리

- 하스스톤에는 고유한 효과를 갖는 카드가 많습니다.
  - 일부 하수인들은 능력을 발동하려면 조건이 필요합니다.
  - 일부 하수인들은 필드에 낼 때 효과를 발동합니다.
  - 일부 하수인들은 죽을 때 효과를 발동합니다.
  - 비밀은 특정 조건에 부합하는 상황이 일어나면 저절로 발동합니다.
- 다양한 카드 효과를 어떻게 처리할 수 있을까요?

카드 효과 처리

- ‘카드 데이터 처리’를 설명하면서 이런 말을 했었습니다.

  (카드의 동작을 구현하는데 필요한 **(거의)** 모든 데이터가 있습니다.)
- 이제 **(거의)**에 대한 이야기를 해봅시다.
- 앞에서 봤던 ‘리로이 젠킨스’ 카드를 다시 한 번 살펴봅시다.

카드 효과 처리


```
{
    "id": "EX1_116",
    "name": "Leeroy Jenkins",
    "text": "Charge. Battlecry: Summon two 1/1 Whelps for your opponent.",
    "attack": 6,
    "cardClass": "NEUTRAL",
    "collectible": true,
    "cost": 5,
    "elite": true,
    "faction": "ALLIANCE",
    "health": 2,
    "mechanics": [
        "BATTLECRY",
        "CHARGE"
    ],
    "rarity": "LEGENDARY",
    "set": "EXPERT1",
    "type": "MINION"
}
```


카드 효과 처리

- ‘카드 데이터 처리’를 설명하면서 이런 말을 했었습니다.

  (카드의 동작을 구현하는데 필요한 **(거의)** 모든 데이터가 있습니다.)
- 이제 **(거의)**에 대한 이야기를 해봅시다.
- 앞에서 봤던 ‘리로이 젠킨스’ 카드를 다시 한 번 살펴봅시다.
- 카드 데이터로는 **카드의 효과를 읽을 수 없습니다.**

카드 효과 처리

- 또한 하스스톤에는 수많은 카드가 존재합니다.
- 여러분이 구현해야 할 카드는 생각보다 많습니다.
- 지금까지 하스스톤에서 출시된 카드의 장수는 1,926장입니다.
  - 정말 1,926장 뿐일까요?

카드 효과 처리


4
Fal'dorei Strider
Battlecry: Shuffle 3 Ambushes into your deck. When drawn, summon a 4/4 Spider.
4
4
Spider Ambush!
Summon a 4/4 Spider. Draw a card. Cast this when drawn.
4
4
Leyline Spider
Beast
4

카드 효과 처리

수집 가능한 카드

수집 불가능한 카드

수집 불가능한 카드

카드 효과 처리

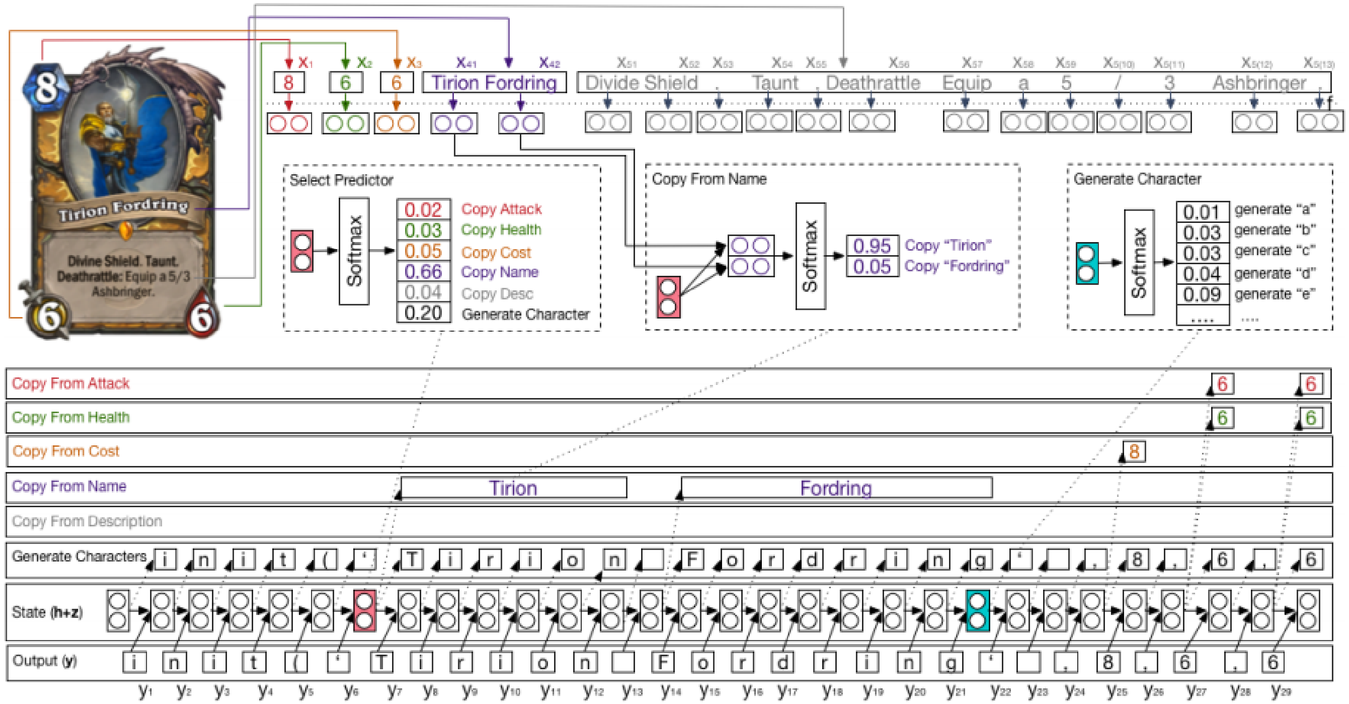
- 또한 하스스톤에는 수많은 카드가 존재합니다.
- 여러분이 구현해야 할 카드는 생각보다 많습니다.
- 지금까지 하스스톤에서 출시된 카드의 장수는 1,926장입니다.
  - 정말 1,926장 뿐일까요?
  - **수집 가능 여부와 관계없이 게임 내에서 사용할 수 있는 카드를 모두 구현해야 합니다.**
  - 그 외 모험 모드에서 사용하는 카드와 내부 테스트용 카드를 합치면 무려 6,086장이나 됩니다.
- 카드의 효과를 쉽게 구현할 방법이 없을까요?

카드 효과를 구현하려면

- 첫번째 방법 : 카드를 읽어 효과를 자동으로 생성하는 방법을 찾아봅니다.

카드 효과를 구현하려면

- Latent Predictor Networks for Code Generation (2016)




```
class MadderBomber(MinionCard):        BLEU = 100.0
  def __init__(self):
    super().__init__("Madder Bomber", 5,
    CHARACTER_CLASS.ALL, CARD_RARITY.RARE,
    battlecry=Battlecry(Damage(1),
    CharacterSelector(players=BothPlayer(),
      picker= RandomPicker(6))))

def create_minion(self, player):§
  return Minion(5, 4)§
```


```
class Preparation(SpellCard):          BLEU = 64.2
  def __init__(self):
    super().__init__("Preparation", 0,
    CHARACTER_CLASS.ROGUE, CARD_RARITY.EPIC,
    target_func=hearthbreaker.targeting.find_minion_spell_target)

  def use(self, player, game):
    super().use(player, game)
    self.target.change_attack(3)
                    player.add_aura(AuraUntil(ManaChange(-3),
                    CardSelector(condition=IsSpell()), SpellCast()))
```

https://arxiv.org/pdf/1603.06744.pdf

카드 효과를 구현하려면

- 첫번째 방법 : 카드를 읽어 효과를 자동으로 생성하는 방법을 찾아봅니다.
  - 선행 연구가 있긴 한데 쉽지 않고 올바르게 생성이 되지 않는 경우도 많습니다. (실패)
- 두번째 방법 : 카드마다 효과를 직접 구현합니다.
  - 정말로 하기 힘들지만 가장 확실한 방법입니다.
  - 다행히 효과가 없는 카드도 존재하기 때문에 6,086장까진 구현하지 않아도 됩니다.

카드 효과의 종류

- 전투의 함성 : 플레이어가 카드를 핸드에서 낼 때 한 번만 발동하는 효과
- 죽음의 메아리 : 하수인이나 무기가 파괴될 때 발동하는 효과

카드 효과의 종류

- 인챈트(Enchant) : 지정된 대상에게 특정 효과를 부여
- 오라(Aura) : 하수인이 필드에 있는 동안 지정된 대상에게 지정된 효과가 유지

카드 효과의 종류

- 트리거(Trigger) : 정해진 조건을 충족할 때 특정 효과가 발동

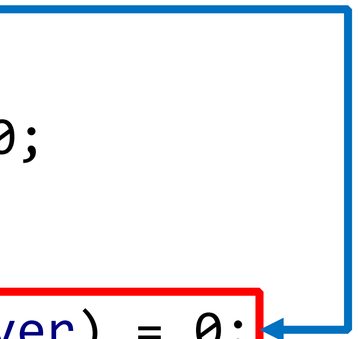
전투의 함성 / 죽음의 메아리 구현

- 태스크를 만들어 처리합니다.
  - 태스크를 만들기 위해서는 먼저 부모 클래스 `ITask`를 상속받아야 합니다.
  - 그리고 순수 가상 함수 `GetTaskID()`와 `Impl()`을 구현해야 합니다.
    - `GetTaskID()` : 태스크 ID를 반환하는 함수
    - `Impl()` : 태스크 로직을 구현하는 함수
  - 구현한 태스크는 `Run()` 또는 `RunMulti()` 함수를 호출해 실행할 수 있습니다.
    - `Run()` : 하나의 태스크를 실행하는 함수
    - `RunMulti()` : 여러 태스크를 연속으로 실행하는 함수

전투의 함성 / 죽음의 메아리 구현

- ITask 클래스

```
class ITask
{
 public:
    TaskStatus Run(Player& player);

    virtual TaskID GetTaskID() const = 0;

 private:
    virtual TaskStatus Impl(Player& player) = 0;
};
```



전투의 함성 / 죽음의 메아리 구현

- Run(), RunMulti() 함수

```
inline TaskStatus Run(Player& player, ITask&& task) {
    return task.Run(player);
}

template <typename... TaskType>
std::vector<TaskStatus> RunMulti(Player& player, TaskType&&... task) {
    std::vector<TaskStatus> metas;
    metas.reserve(sizeof...(task));
    (..., metas.push_back(task.Run(player)));
    return metas;
}
```

전투의 함성 / 죽음의 메아리 구현

- 예제로 카드 ‘방패 올리기’를 구현해 봅시다.
  - 카드를 보고 어떤 태스크를 만들어야 할 지 생각해 봅시다.

전투의 함성 / 죽음의 메아리 구현

- 예제로 카드 '방패 올리기'를 구현해 봅시다.
  - 카드를 보고 어떤 태스크를 만들어야 할 지 생각해 봅시다.
  - 이 카드는 2종류의 태스크가 필요합니다.
    - Gain 5 Armor → `ArmorTask`
    - Draw a card → `DrawTask`
  - `ArmorTask`는 방어도를 얼마나 올릴 것인지 지정합니다.
  - `DrawTask`는 카드를 몇 장이나 드로우할 것인지 지정합니다.

전투의 함성 / 죽음의 메아리 구현

- ArmorTask 클래스 : 방어도를 획득합니다.

```
class ArmorTask : public ITask {
 public:
    explicit ArmorTask(int amount);

    TaskID GetTaskID() const override;

 private:
    TaskStatus Impl(Player& player) override;

    int m_amount = 0;
};
```

전투의 함성 / 죽음의 메아리 구현

- ArmorTask 클래스 : 방어도를 획득합니다.

```
ArmorTask::ArmorTask(int amount) : m_amount(amount) { }

TaskID ArmorTask::GetTaskID() const {
    return TaskID::ARMOR;
}

TaskStatus ArmorTask::Impl(Player& player) {
    player.GetHero()->GainArmor(m_amount);

    return TaskStatus::COMPLETE;
}
```

```
void Hero::GainArmor(int amount) {
    SetArmor(GetArmor() + amount);
}
```

전투의 함성 / 죽음의 메아리 구현

- DrawTask 클래스 : 카드를 드로우합니다.

```
class DrawTask : public Itask {
 public:
    explicit DrawTask(int amount);

    TaskID GetTaskID() const override;

 private:
    TaskStatus Impl(Player& player) override;

    int m_amount = 0;
};
```

전투의 함성 / 죽음의 메아리 구현

- DrawTask 클래스 : 카드를 드로우합니다.

```
DrawTask::DrawTask(int amount) : m_amount(amount) { }

TaskID DrawTask::GetTaskID() const {
    return TaskID::DRAW;
}

TaskStatus DrawTask::Impl(Player& player) {
    for (int i = 0; i < m_amount; ++i)
        Generic::Draw(player, nullptr);
    return TaskStatus::COMPLETE;
}
```

```
Entity* Draw(Player& player, Entity* cardToDraw) {
    if (player.GetDeck().IsEmpty()) {
        int fatigueDamage =
            player.GetHero()->fatigue == 0 ? 1 : player.GetHero()->fatigue + 1;
        player.GetHero()->TakeDamage(*player.GetHero(), fatigueDamage);
        return nullptr;
    }

    Entity* topCard = player.GetDeck().GetTopCard();
    if (topCard == nullptr) {
        return nullptr;
    }
    Entity* entity = &player.GetDeck().RemoveCard(*topCard);

    AddCardToHand(player, entity);
    return entity;
}
```

```
void AddCardToHand(Player& player, Entity* entity) {
    if (player.GetHand().IsFull())
    {
        player.GetGraveyard().AddCard(*entity);
        return;
    }

    player.GetHand().AddCard(*entity);
}
```

전투의 함성 / 죽음의 메아리 구현

- 구현한 태스크를 통해 카드 효과를 등록합니다.

```
// ----------------------------------- SPELL - WARRIOR
// [EX1_606] Shield Block - COST:3
// - Faction: Neutral, Set: Core, Rarity: Free
// ---------------------------------------------------
// Text: Gain 5 Armor.
//       Draw a card.
// ---------------------------------------------------
power.ClearData();
power.AddPowerTask(new ArmorTask(5));
power.AddPowerTask(new DrawTask(1));
cards.emplace("EX1_606", power);
```

영웅/하수인 공격 구현

- 공격하기 전에
  - 영웅이나 하수인이 공격할 수 있는지 확인합니다.
  - 공격할 대상이 유효한지 확인합니다.
- 공격할 때
  - 빙결, 독성, 은신, 돌진 등의 특수 능력이 있는지 확인합니다.
  - 피해를 주고 난 뒤 남은 체력에 따라 영웅/하수인을 파괴합니다.

```
void Attack(Player& player, Character* source, Character* target) {
    // Check source can attack and target is valid
    if (!source->CanAttack() ||
        !source->IsValidCombatTarget(*player.opponent, target))
        return;

    // Activate attack trigger
    player.GetGame()->triggerManager.OnAttackTrigger(&player, source);
    player.GetGame()->ProcessTasks();

    // Set game step to MAIN_COMBAT
    player.GetGame()->step = Step::MAIN_COMBAT;

    ...
```

```cpp
bool Character::CanAttack() {
    // If the value of attack is 0, returns false
    if (GetAttack() == 0)
        return false;

    // If the character is frozen, returns false
    if (GetGameTag(GameTag::FROZEN) == 1)
        return false;

    // If the character is exhausted, returns false
    if (GetExhausted())
        return false;

    //! If the character can't attack, returns false
    if (GetGameTag(GameTag::CANT_ATTACK) == 1)
        return false;

    return true;
}
```

```
bool Character::IsValidCombatTarget(Player& opponent, Character* target) const {
    auto targets = GetValidCombatTargets(opponent);
    if (std::find(targets.begin(), targets.end(), target) == targets.end())
        return false;

    const Hero* hero = dynamic_cast<Hero*>(target);
    return !(hero != nullptr &&
             hero->GetGameTag(GameTag::CANNOT_ATTACK_HEROES) == 1);
}
```

```cpp
std::vector<Character*> Character::GetValidCombatTargets(Player& opponent) const {
    bool isExistTauntInField = false;
    std::vector<Character*> targets, targetsHaveTaunt;

    for (auto& minion : opponent.GetField().GetAllMinions()) {
        if (minion->GetGameTag(GameTag::STEALTH) == 0) {
            if (minion->GetGameTag(GameTag::TAUNT) == 1) {
                isExistTauntInField = true;
                targetsHaveTaunt.emplace_back(minion);
                continue;
            }
            if (!isExistTauntInField)
                targets.emplace_back(minion);
        }
    }

    if (isExistTauntInField)
        return targetsHaveTaunt;

    if (GetGameTag(GameTag::CANNOT_ATTACK_HEROES) == 0 &&
        opponent.GetHero()->GetGameTag(GameTag::IMMUNE) == 0 &&
        opponent.GetHero()->GetGameTag(GameTag::STEALTH) == 0)
        targets.emplace_back(opponent.GetHero());

    return targets;
}
```

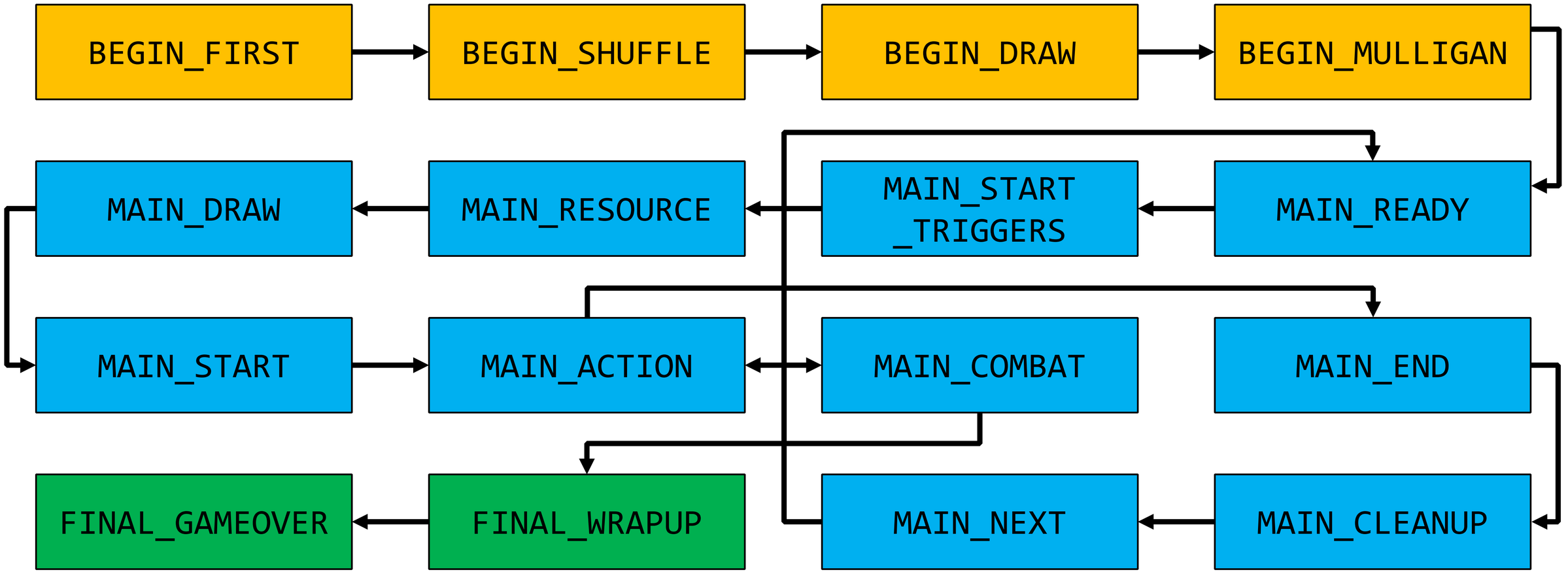
```
void Attack(Player& player, Character* source, Character* target) {
    ...

    // Get attack of source and target
    const int targetAttack = target->GetAttack();
    const int sourceAttack = source->GetAttack();

    // Take damage to target
    const int targetDamage = target->TakeDamage(*source, sourceAttack);
    const bool isTargetDamaged = targetDamage > 0;

    // Freeze target if attacker is freezer
    if (isTargetDamaged && source->GetGameTag(GameTag::FREEZE) == 1)
        target->SetGameTag(GameTag::FROZEN, 1);

    // Destroy target if attacker is poisonous
    if (isTargetDamaged && source->GetGameTag(GameTag::POISONOUS) == 1)
        target->Destroy();
```

```
// Ignore damage from defenders with 0 attack
if (targetAttack > 0)
{
    // Take damage to source
    const int sourceDamage = source->TakeDamage(*target, targetAttack);
    const bool isSourceDamaged = sourceDamage > 0;

    // Freeze source if defender is freezer
    if (isSourceDamaged && target->GetGameTag(GameTag::FREEZE) == 1)
        source->SetGameTag(GameTag::FROZEN, 1);

    // Destroy source if defender is poisonous
    if (isSourceDamaged && target->GetGameTag(GameTag::POISONOUS) == 1)
        source->Destroy();
}

// Remove stealth ability if attacker has it
if (source->GetGameTag(GameTag::STEALTH) == 1)
    source->SetGameTag(GameTag::STEALTH, 0);
```

```
// Remove durability from weapon if hero attack
Hero* hero = dynamic_cast<Hero*>(source);
if (hero != nullptr && hero->weapon != nullptr &&
    hero->weapon->GetGameTag(GameTag::IMMUNE) == 0)
{
    int prevValue = hero->weapon->GetDurability();
    hero->weapon->SetDurability(prevValue - 1);

    // Destroy weapon if durability is 0
    if (hero->weapon->GetDurability() == 0)
        hero->RemoveWeapon();
}

// Increase the number of attacked
source->SetNumAttacksThisTurn(source->GetNumAttacksThisTurn() + 1);
```

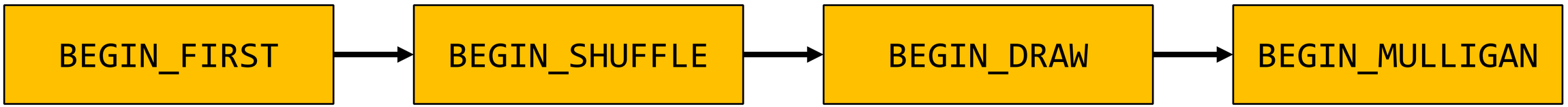
```
    // Check source is exhausted
    if ((source->GetNumAttacksThisTurn() >= 1 &&
         source->GetGameTag(GameTag::WINDFURY) == 0) ||
        (source->GetNumAttacksThisTurn() >= 2 &&
         source->GetGameTag(GameTag::WINDFURY) == 1))
        source->SetExhausted(true);

    // Process destroy and update aura
    player.GetGame()->ProcessDestroyAndUpdateAura();

    // Set game step to MAIN_ACTION
    player.GetGame()->step = Step::MAIN_ACTION;
}
```

하스스톤의 게임 진행



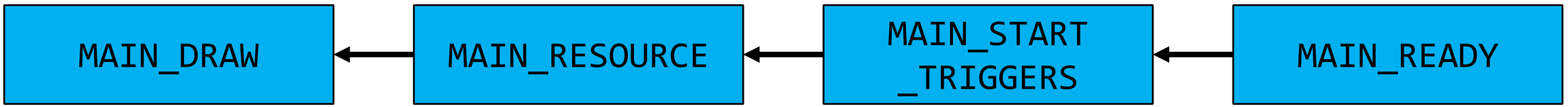
하스스톤의 게임 진행

- BEGIN_FIRST : 게임을 시작할 때 처리해야 할 로직을 수행합니다.
- BEGIN_SHUFFLE : 덱에 있는 카드들을 무작위로 섞습니다.
- BEGIN_DRAW : 카드를 뽑습니다. (선 플레이어 : 3장, 후 플레이어 : 4장 + 동전)
- BEGIN_MULLIGAN : 마음에 들지 않는 카드를 선택해 덱에 넣고 다시 뽑습니다.



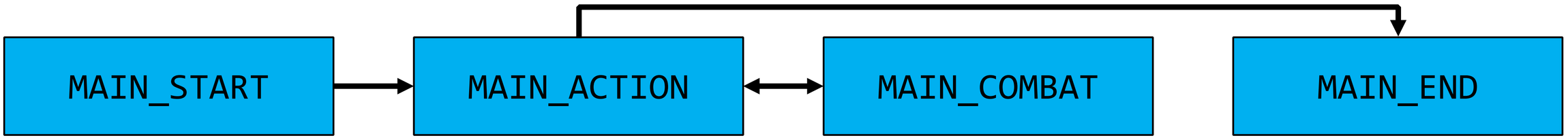
하스스톤의 게임 진행

- MAIN_READY : 영웅, 무기, 하수인의 공격 가능 횟수를 초기화합니다.
- MAIN_START_TRIGGERS : 턴을 시작할 때 동작하는 트리거를 호출합니다.
- MAIN_RESOURCE : 마나 수정을 1개 추가하고 모두 채웁니다.
- MAIN_DRAW : 덱에서 카드 1장을 뽑습니다.



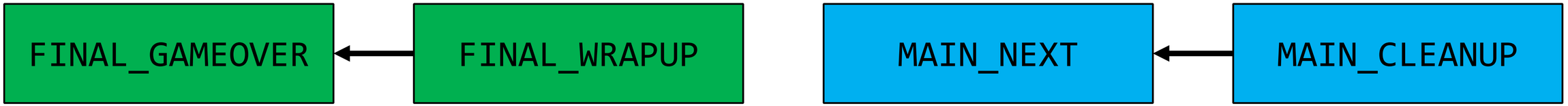
하스스톤의 게임 진행

- MAIN_START : 플레이어의 턴을 시작할 때 처리해야 할 로직을 수행합니다.
- MAIN_ACTION : 플레이어의 행동을 처리합니다. (예 : 카드를 낸다, 턴을 종료한다)
- MAIN_COMBAT : 하수인이 상대편의 하수인이나 영웅을 공격합니다.
- MAIN_END : 플레이어의 턴이 끝날 때 처리해야 할 로직을 수행합니다.



하스스톤의 게임 진행

- MAIN_CLEANUP : 한 턴만 지속되는 효과를 제거하는 등의 로직을 수행합니다.
- MAIN_NEXT : 상대 플레이어에게 턴을 넘깁니다.
- FINAL_WRAPUP : 게임의 승패를 처리합니다.
- FINAL_GAMEOVER : 게임과 관련된 데이터를 정리하고 종료합니다.



하스스톤의 게임 진행

- 예제 코드 : BEGIN_SHUFFLE 단계

```
void Game::BeginShuffle() {
    if (m_gameConfig.doShuffle) {
        GetPlayer1().GetDeck().Shuffle();
        GetPlayer2().GetDeck().Shuffle();
    }

    nextStep = Step::BEGIN_DRAW;
    if (m_gameConfig.autoRun) {
        GameManager::ProcessNextStep(*this, nextStep);
    }
}
```

하스스톤의 게임 진행

- 예제 코드 : MAIN_START_TRIGGERS 단계

```
void Game::MainStartTriggers() {
    triggerManager.OnStartTurnTrigger(&GetCurrentPlayer(), nullptr);
    ProcessTasks();
    ProcessDestroyAndUpdateAura();

    nextStep = Step::MAIN_RESOURCE;
    if (m_gameConfig.autoRun) {
        GameManager::ProcessNextStep(*this, nextStep);
    }
}
```

NEXON DEVELOPERS CONFERENCE

4. 하스스톤 강화학습 환경 개발

- 정책 클래스 구현
- PyTorch C++ API 연동

정책 클래스 구현

- 지금까지 우리는 하스스톤 게임을 만들었습니다.

  하지만 우리의 목표는 AI가 하스스톤 게임을 하도록 만드는 것입니다.
- 이를 위해 AI가 게임을 할 수 있게 만들어야 합니다.
  - AI가 행동을 하기 위한 통로를 만들어야 합니다.
  - AI가 결정한 값을 게임에 전달할 수 있어야 합니다.
- 두 가지 조건을 해결하려면 정책 클래스를 만들어야 합니다.
  - 테스트를 위해 임의의 행동을 하는 정책 클래스를 구현해 봅시다.

정책 클래스 구현

- IPolicy 인터페이스

```
class IPolicy {
 public:
    //! Gets next behavior from given \p game.
    virtual TaskMeta Next(const Game& game) = 0;

    //! Gets proper requirement with given \p player and \p id.
    virtual TaskMeta Require(Player& player, TaskID id) = 0;

    //! Notify serialized data to IPolicy.
    virtual void Notify(const TaskMeta& meta) = 0;
};
```

정책 클래스 구현

- `IPolicy` 인터페이스
  - Next() 함수 : 게임에서 다음에 수행할 행동을 반환합니다.

    (예 : 카드 내기, 하수인으로 공격하기, 카드 멀리건, 턴 종료)
  - Require() 함수 : 지금 하려는 행동에 대해 필요한 정보를 받습니다.

    (예 : 하수인으로 공격하려면 현재 필드가 어떤 상태인지 알아야 합니다.)
  - Notify() 함수 : 게임에서 발생한 부가 정보를 직렬화해서 보냅니다.

    (예 : 플레이어의 핸드에 카드가 10장이 넘어서 카드가 탔다는 정보를 보냅니다.)

PyTorch C++ API 연동

- PyTorch
  - https://pytorch.org/
  - Python 기반의 오픈 소스 머신 러닝 라이브러리입니다.
  - C++ API(LibTorch) 와 Python API를 동시 지원합니다.
  - C++ 기반의 머신러닝 모델 설계에 용이합니다.
- 하스스톤의 강화학습 환경을 구성하기 위해 PyTorch C++ API(LibTorch)를 사용했습니다.

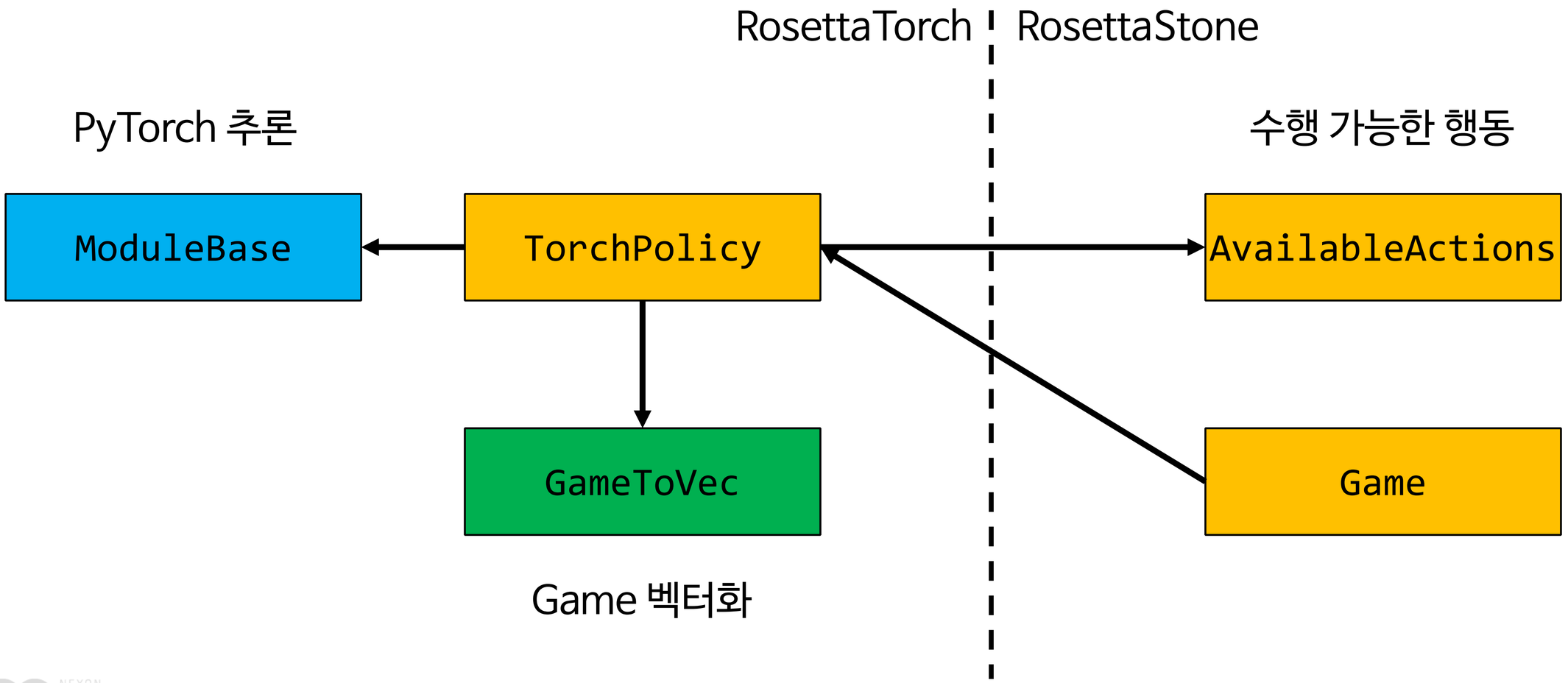
RosettaTorch

- PyTorch C++ API 기반 정책을 위한 프로젝트
- 게임 상태를 받아 현재 취할 수 있는 행동을 선별합니다.
- 게임의 상태 정보를 추합해 벡터로 표현합니다.
- 현재 취할 수 있는 행동, 게임 벡터를 기반으로 다음 행동을 추론합니다.
- 다음 행동을 정책 인터페이스를 통해 게임으로 전달합니다.

RosettaTorch

- 프로젝트 구성
  - `TorchPolicy` : PyTorch 기반의 정책 클래스
  - `AvailableActions` : 현재 게임 상태에서 취할 수 있는 행동 벡터
  - `ModuleBase` : PyTorch 기반의 딥러닝 모듈
  - `GameToVec` : 게임 상태를 벡터로 취합하는 인터페이스

RosettaTorch



RosettaTorch

- TorchPolicy 클래스

```
class TorchPolicy : public BasicPolicy {
 public:
    //! Constructs torch based policy with given module.
    //! \param module The module base for torch deep learning policy.
    //! \param gameToVec The generator for converting game to vector.
    TorchPolicy(std::shared_ptr<ModuleBase> module, GameToVec* gameToVec);

    //! Gets next behavior with random actions.
    TaskMeta Next(const Game& game) override;
};
```

RosettaTorch

- `ModuleBase` 클래스
  - `torch::nn::Module`를 상속 받은 모듈 클래스 입니다.
  - 현재 발생 가능한 행동 벡터와 게임 상태 벡터를 입력으로 받습니다.
  - 행동 벡터와 같은 크기의 확률 벡터를 반환합니다.
  - `TorchPolicy`는 이중 가장 큰 확률이 부여된 행동을 게임에 반환합니다.

RosettaTorch

- ModuleBase 클래스

```
class ModuleBase : public torch::nn::Module {
 public:
    //! Forwarding tensors to generate decision.
    //! \param available The available actions.
    //! \param context The game context encoded by GameToVec.
    //! \return Probability to make decision based on available actions.
    virtual torch::Tensor forward(torch::Tensor available,
                                  torch::Tensor context);
};
```

RosettaTorch

- GameToVec 클래스
  - 게임 상태를 추합하여 벡터로 만드는 인터페이스입니다.
  - 상태 벡터의 크기와 형태는 무관합니다.
  - 추합된 벡터는 발생 가능한 액션 목록과 함께 추론에 이용됩니다.

RosettaTorch

- GameToVec 클래스

```
class GameToVec {
 public:
    //! Generate torch tensor from game context.
    //! \param game The game context.
    //! \return The encoded torch tensor from \p game.
    virtual torch::Tensor GenerateTensor(const Game& game);
};
```

GameToVec

- 게임 실행 중 알 수 있는 모든 정보를 벡터로 만들어 봅시다.
- 게임 실행 중 알 수 있는 정보에는 어떤 것들이 있을까요?


고대의 부스트라즈
1/10
바탕 화면에 스크린샷이 저장되었습니다.
32
0 3
2 3
19 19
18 18
17 13
16 12
15 11
14 10
턴 종료
2
50
유틸
6/10
9
오전 12:21
60

GameToVec

- 게임 실행 중 알 수 있는 모든 정보를 벡터로 만들어 봅시다.
- 게임 실행 중 알 수 있는 정보에는 어떤 것들이 있을까요?
  - 나와 상대방의 손에 있는 카드 장 수
  - 나와 상대방의 덱에 남아있는 카드 장 수
  - 나와 상대방의 필드에 있는 하수인 정보
  - 나의 손에 있는 카드들의 정보
  - 나의 덱에 남아 있는 카드들의 정보

GameToVec

- 나와 상대방의 손과 덱에 있는 카드 장 수
  - Sparse한 값을 MAX 값으로 나눠서 0~1 사이의 값을 갖도록 정규화합니다.
  - 상대방 손에 있는 카드 개수

    ```
    opPlayer.GetHand().GetNumOfCards() / HAND_SIZE;
    ```
  - 상대방 덱에 있는 카드 개수

    ```
    opPlayer.GetDeck().GetNumOfCards() / MAX_DECK_SIZE;
    ```
  - 내 덱에 남은 카드 개수

    ```
    curPlayer.GetDeck().GetNumOfCards() / MAX_DECK_SIZE;
    ```

GameToVec

- 나와 상대방의 필드 그리고 나의 손, 덱에 있는 카드의 정보
  - 마나 비용, 공격력, 체력 값은 0~1 사이의 값을 갖도록 정규화합니다.

    이때 공격력과 체력의 타입은 int32인데 일정 값보다 크다면 비슷한 상황이라 판단해

    CLIP_NORM(64) 이상은 1, 이하는 64로 나눈 값을 채택합니다.
  - 마나 비용 : `cost / MANA_UPPER_LIMIT;`
  - 공격력 : `(attack >= CLIP_NORM) ? 1.0f : attack / CLIP_NORM;`
  - 체력 : `(health > CLIP_NORM) ? 1.0f : health / CLIP_NORM;`

GameToVec

- 나와 상대방의 필드 그리고 나의 손, 덱에 있는 카드의 정보
  - 효과와 관련된 정보는 어떻게 벡터로 표현할까요?

    (`Aura`, `Enchant`, `Deathrattle`, `Power` 등)
  - 각 효과 정보를 인덱싱한 뒤 임베딩 룩업 테이블

    (Embedding Lookup Table)에서 벡터를 검색합니다.

GameToVec

- 나와 상대방의 필드 그리고 나의 손, 덱에 있는 카드의 정보
  - 예를 들어, `Effect`에는 변수로 `GameTag`와 `EffectOperator`가 있습니다.

    이때 `GameTag::ATK`가 1이고 `EffectOperator::SET`이 30이라면

    1 * 3 = 3, 즉 3번 인덱스를 통해 3번 벡터를 찾을 수 있습니다.

Lookup table

Index 0	Vector 0
Index 1	Vector 1
Index 2	Vector 2
Index 3	Vector 3
…	…
Index N	Vector N

GameToVec

- 벡터의 전체 크기
  - `Ability2Vec` : 5(aura) + 4(enchant) + 8(deathrattle) + 8(power) = m = 25
  - `Card2Vec` : 1(cost) + 1(health) + 1(attack) + m(ability) = n = 28
  - `Game2Vec` : 1(카드 수) + 1(카드 수) + 1(카드 수) + $n * 7 + n * 7 + n * 10$

    $= 3 + 24 * n$(card) = 675
- 현재 총 675차원의 벡터로 게임을 표현하고 있습니다.

PyTorch C++ API 연동

- RosettaTorch를 통해 PyTorch C++ API 기반의 정책을 구현했습니다.
- 이제 여러 딥러닝 알고리즘을 활용해 하스스톤의 정책을 결정해보고,
  그 결과들을 데이터로 알고리즘을 학습할 환경이 완성되었습니다.
- 게임 상태를 추합하는 `GameToVec`, 실제 추론을 진행하는 `ModuleBase`를
  수정해 볼 수 있으며, 그 외 학습 기법을 적용해 볼 수 있게 되었습니다.

NEXON
DEVELOPERS
CONFERENCE

5. 개발 진행 상황

- 프로젝트 소개
- 카드 구현 상황
- 강화학습 환경 개발 상황
- 앞으로의 계획

RosettaStone

Hearthstone simulator using C++ with some reinforcement learning

- https://github.com/utilForever/RosettaStone
- 하스스톤 시뮬레이터 + 강화학습 환경 제공
- 라이선스 : AGPLv3
- 작성 언어 : C++17
- 지원 컴파일러
  - g++ (7.0 버전 이상)
  - clang (5 버전 이상)
  - Microsoft Visual C++ (2017 버전 이상)

프로젝트 구조

- RosettaStone : 하스스톤 시뮬레이터 라이브러리
- RosettaConsole : 하스스톤 게임을 해볼 수 있는 콘솔 프로그램
- RosettaGUI : 하스스톤 게임을 해볼 수 있는 GUI 프로그램
- RosettaTorch : 강화학습을 위해 정책을 구현할 수 있는 라이브러리
- RosettaTool : 카드 구현 목록을 쉽게 확인할 수 있는 툴 프로그램

카드 구현 상황

- Basic & Classic
  - 89% Basic (119 of 133 Cards) + 9% Classic (23 of 237 Cards)
  - 9% Hall of Fame (2 of 22 Cards)
- Expansions
  - 0% Rise of Shadows (0 of 135 cards)
  - 0% Rastakhan's Rumble (0 of 135 Cards)
  - 0% The Boomsday Project (0 of 135 Cards)
  - 0% The Witchwood (0 of 135 Cards)

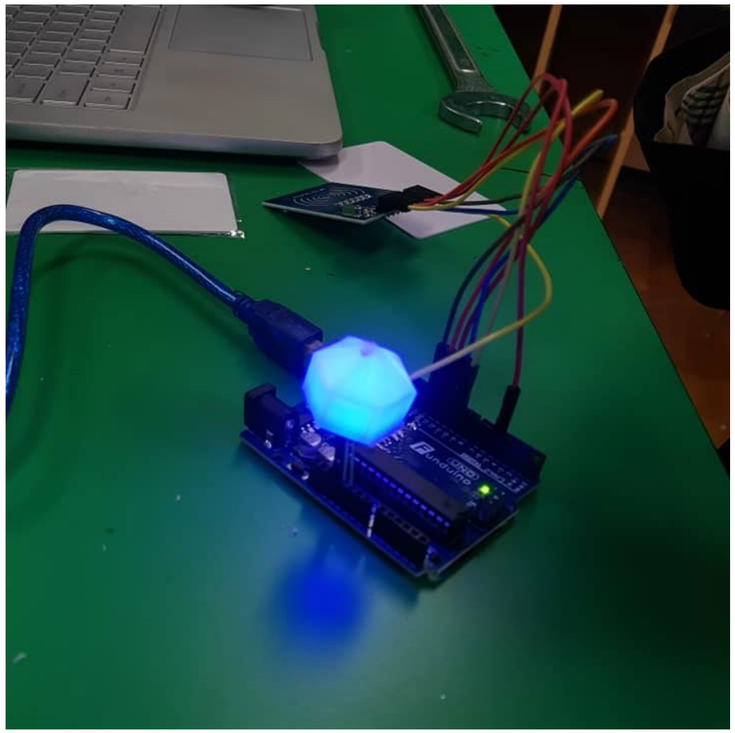
강화학습 환경 개발 상황

- 정책 클래스 구현 완료
  - `IPolicy`, `BasicPolicy`, `RandomPolicy`
- PyTorch C++ API 연동 완료
  - RosettaTorch 프로젝트 (`TorchPolicy`)
- 정보 전달 함수 구현 마무리 중
  - `AvailableActions()`, `GameToVec()`
- DQN을 사용한 예제 준비중 (5월 중 공개할 예정)

앞으로의 계획

- 모든 카드를 구현할 예정입니다.
  - 우선적으로 오리지널 카드와 정규전에서 사용하는 카드를 구현합니다.
  - 이후 야생전에서 사용하는 카드를 차례대로 구현할 예정입니다.
- 콘솔 및 GUI 프로그램을 개선해 AI와 대전할 수 있게 만들 예정입니다.
- 다른 언어에서 사용할 수 있도록 API를 제공할 예정입니다.
  - 우선적으로 Python을 생각하고 있습니다.
- “RealStone” 팀과 협력해 실제 기기로 대전할 수 있게 만들 예정입니다.

RealStone 프로젝트



NEXON
DEVELOPERS
CONFERENCE

6. 정리

정리

- 게임에서 API를 제공한다면 강화학습을 편하게 연구할 수 있습니다.
  하지만 제공하지 않는다면 힘들지만 게임부터 직접 만들어야 합니다.
- 게임을 만드는 작업이 끝나면 강화학습을 위한 작업을 진행해야 합니다.
  행동을 결정할 수 있도록 AI에 게임 정보를 전달하는 함수와
  AI가 결정한 행동을 게임에 전달하는 함수를 구현해야 합니다.
- 이 때 Tensorflow나 PyTorch와 연동하기 위해 정보를 변환해야 합니다.
- 이 작업까지 끝나면 비로소 강화학습 연구를 할 수 있게 됩니다.

감사합니다.

RosettaStone 프로젝트를 함께 발전시켜나갈 분들을 찾고 있습니다.

카드 구현이나 강화학습 연구, 프로그램 구현에 관심이 있다면 꼭 연락주세요!

NEXON DEVELOPERS CONFERENCE

Bonus) 다루지 못했던 내용

- better-enums의 문제점
  - 대안 : X Macro와 함수 템플릿 특수화로 해결하기
- 카드 동작 테스트
- 인챈트 / 오라 / 트리거 구현
- 정책 클래스 구현

better-enums의 문제점

- 열거체 GameTag는 하스스톤 게임과 관련된 다양한 데이터를 저장합니다.
  - 공격력, 체력 등 직접적인 정보 뿐만 아니라 어빌리티, 특징 등 간접적인 정보도 저장합니다.
  - 최신 확장팩을 기준으로 총 455개의 게임 태그 멤버가 있습니다.
- 문제는 better-enums 라이브러리가 매크로 기반 구현체라는 점입니다.
  - 매크로에서 사용 가능한 인수/매개 변수의 개수에는 제한이 있습니다.

    (C++ 표준 : 256개, Visual C++ 컴파일러 : 127개)
  - 따라서 열거체 GameTag는 better-enums를 사용할 수 없습니다.
  - 다른 방법이 없을까요?

https://docs.microsoft.com/en-us/cpp/cpp/compiler-limits?view=vs-2017

X Macro

- 코드 반복을 최소화하고 데이터와 코드를 분리해서 처리할 수 있는 전처리 기법
- 크게 두 부분으로 구성되어 있습니다.
  - 데이터를 정의하는 부분
  - 데이터를 확장하는 부분
- X Macro 기법을 활용해 열거체 타입과 문자열 변환 함수를 만들어 봅시다.

https://en.wikipedia.org/wiki/X_Macro

X Macro와 함수 템플릿 특수화로 해결하기

- 1. 열거체를 저장할 파일을 만듭니다. (예 : `Fruit.def`)
- 2. 열거체 멤버들을 X(이름) 형태로 정의합니다.

```
X(APPLE)
X(BANANA)
X(ORANGE)
X(MANGO)
X(GRAPE)
X(PEAR)
```

X Macro와 함수 템플릿 특수화로 해결하기

- 3. 열거체와 문자열 배열을 선언할 헤더 파일을 만듭니다.
- 4. 열거체와 문자열 배열을 다음과 같이 선언합니다.

```
enum class Fruit
{
#define X(a) a,
#include "Fruit.def"
#undef X
};
```

```
const std::string FRUIT_STR[] = {
#define X(a) #a,
#include "Fruit.def"
#undef X
};
```

X Macro와 함수 템플릿 특수화로 해결하기

- 5. 문자열과 열거체를 변환할 함수 템플릿

  StrToEnum과 EnumToStr을 선언합니다.

```
template <class T>
T StrToEnum(std::string_view);
template <class T>
std::string_view EnumToStr(T);
```

X Macro와 함수 템플릿 특수화로 해결하기

- 6. 함수 템플릿 특수화를 위한 매크로를 정의합니다.

```
#define STR2ENUM(TYPE, ARRAY)                                          \
    template <>                                                        \
    inline TYPE StrToEnum<TYPE>(std::string_view str) {                \
        for (int i = 0; i < (sizeof(ARRAY) / sizeof(ARRAY[0])); ++i)   \
            if (ARRAY[i] == str)                                       \
                return TYPE(i);                                        \
                                                                       \
        return TYPE(0);                                                \
    }
```

X Macro와 함수 템플릿 특수화로 해결하기

- 6. 함수 템플릿 특수화를 위한 매크로를 정의합니다.

```
#define ENUM2STR(TYPE, ARRAY)                                  \
    template <>                                                \
    inline std::string_view EnumToStr<TYPE>(TYPE v) { \
        return ARRAY[static_cast<int>(v)];                     \
    }

#define ENUM_AND_STR(TYPE, ARRAY) \
    STR2ENUM(TYPE, ARRAY)         \
    ENUM2STR(TYPE, ARRAY)
```

X Macro와 함수 템플릿 특수화로 해결하기

- 7. ENUM_AND_STR 매크로를 사용해 변환 함수를 특수화합니다.

```
ENUM_AND_STR(Fruit, FRUIT_STR)
```

X Macro와 함수 템플릿 특수화로 해결하기

- 8. 이제 매크로 인수/매개변수 개수에 제한받지 않고 사용할 수 있습니다.

```
int main()
{
    Fruit fruit = StrToEnum<Fruit>("MANGO");
    std::cout << EnumToStr(fruit) << std::endl;

    fruit = Fruit::APPLE;
    std::cout << EnumToStr(fruit) << std::endl;
}
```

카드 동작 테스트

- 카드 효과를 구현했다면 의도한 대로 동작하는지 테스트해봐야 합니다.
- 카드의 동작을 확인하기 위한 시나리오를 작성합니다.
  - 1. 플레이어 1의 손패에서 카드 '방패 올리기'를 냅니다.
  - 2. 플레이어 1의 영웅 방어도가 5 증가했는지 확인합니다.
  - 3. 플레이어 1의 손패에서 카드 장 수가 그대로인지 확인합니다.

    (카드 '방패 올리기'를 내서 1장 감소, 카드를 드로우해서 1장 증가)

카드 동작 테스트

- 카드 '방패 올리기' 동작 테스트

```
TEST(CoreCardsGen, EX1_606) {
    GameConfig config;
    config.player1Class = CardClass::WARRIOR;
    config.player2Class = CardClass::WARLOCK;
    config.startPlayer = PlayerType::PLAYER1;
    config.doFillDecks = true;
    config.autoRun = false;

    Game game(config);
    game.StartGame();
    game.ProcessUntil(Step::MAIN_START);
```

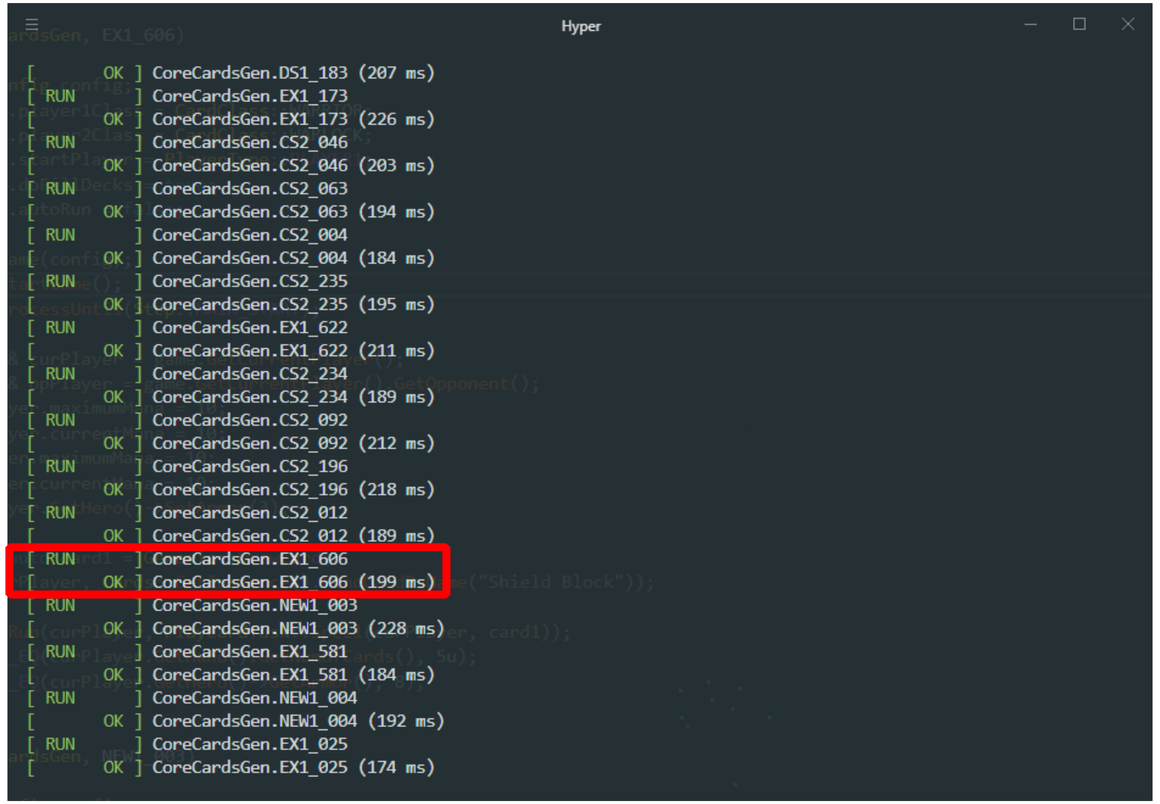
카드 동작 테스트

- 카드 '방패 올리기' 동작 테스트

```
    Player& curPlayer = game.GetCurrentPlayer();
    Player& opPlayer = game.GetCurrentPlayer().GetOpponent();
    curPlayer.maximumMana = curPlayer.currentMana = 10;
    opPlayer.maximumMana = opPlayer.currentMana = 10;

    const auto card1 = Generic::DrawCard(
        curPlayer, Cards::GetInstance().FindCardByName("Shield Block"));
    Task::Run(curPlayer, PlayCardTask::Spell(curPlayer, card1));
    EXPECT_EQ(curPlayer.GetHand().GetNumOfCards(), 5u);
    EXPECT_EQ(curPlayer.GetHero()->GetArmor(), 5);
}
```

카드 동작 테스트

- 단위 테스트를 실행해 동작에 이상이 없는지 확인합니다.



인챈트 구현

- AddEnchantmentTask를 통해 인챈트 효과를 추가할 수 있습니다.
- 이때 주의할 점은 인챈트 카드가 따로 존재한다는 점입니다.

  (주문 카드의 ID가 CS2_092라면 인챈트 카드의 ID는 CS2_092e)
- 예제로 카드 '왕의 축복'을 살펴봅시다.

```
{
    "artist": "Lucas Graciano",
    "cardClass": "PALADIN",
    "collectible": true,
    "cost": 4,
    "dbfId": 943,
    "howToEarn": "Unlocked at Level 10.",
    "howToEarnGolden": "Unlocked at Level 49.",
    "id": "CS2_092",
    "name": "Blessing of Kings",
    "playRequirements": {
        "REQ_MINION_TARGET": 0,
        "REQ_TARGET_TO_PLAY": 0
    },
    "rarity": "FREE",
    "set": "CORE",
    "text": "Give a minion +4/+4. <i>(+4 Attack/+4 Health)</i>",
    "type": "SPELL"
},
```


```
{
    "cardClass": "PALADIN",
    "dbfId": 803,
    "id": "CS2_092e",
    "name": "Blessing of Kings",
    "set": "CORE",
    "text": "+4/+4.",
    "type": "ENCHANTMENT"
},
```

인챈트 구현

- AddEnchantmentTask를 통해 인챈트 카드를 사용한다고 알려줍니다.

```
// ----------------------------------- SPELL - PALADIN
// [CS2_092] Blessing of Kings - COST:4
// - Faction: Neutral, Set: Core, Rarity: Free
// --------------------------------------------------
// Text: Give a minion +4/+4. <i>(+4 Attack/+4 Health)</i>
// --------------------------------------------------
// PlayReq: REQ_TARGET_TO_PLAY = 0, REQ_MINION_TARGET = 0
// --------------------------------------------------
power.ClearData();
power.AddPowerTask(new AddEnchantmentTask("CS2_092e", EntityType::TARGET));
cards.emplace("CS2_092", power);
```

```
TaskStatus AddEnchantmentTask::Impl(Player& player) {
    Card enchantmentCard = Cards::FindCardByID(m_cardID);
    if (enchantmentCard.id.empty())
        return TaskStatus::STOP;

    auto entities = IncludeTask::GetEntities(m_entityType, player, m_source, m_target);
    Power power = Cards::FindCardByID(m_cardID).power;

    for (auto& entity : entities) {
        const auto enchantment =
            Enchantment::GetInstance(player, enchantmentCard, entity);

        if (power.GetAura().has_value())
            power.GetAura().value().Activate(*enchantment);

        if (power.GetTrigger().has_value())
            power.GetTrigger().value().Activate(*enchantment);

        if (power.GetEnchant().has_value()) {
            const auto taskStack = player.GetGame()->taskStack;
            power.GetEnchant().value().ActivateTo(entity, taskStack.num, taskStack.num1);
        }
    }

    return TaskStatus::COMPLETE;
}
```

```
void Enchant::ActivateTo(Character* character) {
    for (auto& effect : effects)
        effect.Apply(character);
}
```

인챈트 구현

- 인챈트 카드도 등록해줍니다.

```
// ---------------------------------- ENCHANTMENT - PALADIN
// [CS2_092e] Blessing of Kings (*) - COST:0
// - Set: Core
// --------------------------------------------------------
// Text: +4/+4.
// --------------------------------------------------------
power.ClearData();
power.AddEnchant(Enchants::GetEnchantFromText("CS2_092e"));
cards.emplace("CS2_092e", power);
```

```cpp
Enchant Enchants::GetEnchantFromText(const std::string& cardID) {
    std::vector<Effect> effects;
    bool isOneTurn = false;

    static std::regex attackHealthRegex("\\+([[:digit:]]+)/\\+([[:digit:]]+)");
    static std::regex attackRegex("\\+([[:digit:]]+) Attack");
    static std::regex healthRegex("\\+([[:digit:]]+) Health");

    const std::string text = Cards::FindCardByID(cardID).text;
    std::smatch values;

    if (std::regex_search(text, values, attackHealthRegex)) {
        effects.emplace_back(Effects::AttackN(std::stoi(values[1].str())));
        effects.emplace_back(Effects::HealthN(std::stoi(values[2].str())));
    }
    else if (std::regex_search(text, values, attackRegex)) {
        effects.emplace_back(Effects::AttackN(std::stoi(values[1].str())));
    }
    else if (std::regex_search(text, values, healthRegex)) {
        effects.emplace_back(Effects::HealthN(std::stoi(values[1].str())));
    }

    return Enchant(effects, false, isOneTurn);
}
```

오라 구현

- 오라를 구현할 때는 활성화/비활성화 시기를 고려해야 합니다.
  - 하수인이 필드에 있는 동안 활성화됩니다.
  - 하수인이 죽으면 비활성화됩니다.
- 예제로 카드 ‘스톰윈드 용사’를 살펴봅시다.

오라 구현

- AddAura를 통해 오라를 등록합니다.

```
// ------------------------------------ MINION - NEUTRAL
// [CS2_222] Stormwind Champion - COST:7 [ATK:6/HP:6]
// - Faction: Alliance, Set: Core, Rarity: Free
// ---------------------------------------------------
// Text: Your other minions have +1/+1.
// ---------------------------------------------------
// GameTag:
// - AURA = 1
// ---------------------------------------------------
power.ClearData();
power.AddAura(Aura("CS2_222o", AuraType::FIELD_EXCEPT_SOURCE));
cards.emplace("CS2_222", power);
```

```
void PlayMinion(Player& player, Minion* minion, Character* target, int fieldPos) {
    // Add minion to battlefield
    player.GetField().AddMinion(*minion, fieldPos);

    // Apply card mechanics tags
    for (const auto tags : minion->card.gameTags)
        minion->SetGameTag(tags.first, tags.second);

    // Process power tasks
    for (auto& powerTask : minion->card.power.GetPowerTask()) {
        if (powerTask == nullptr)
            continue;

        powerTask->SetSource(minion);
        powerTask->SetTarget(target);
        powerTask->Run(player);
    }

    player.GetGame()->ProcessDestroyAndUpdateAura();
}
```

```
void Battlefield::AddMinion(Minion& minion, std::size_t pos) {
    m_minions.at(pos) = &minion;
    ++m_numMinion;

    if (minion.GetGameTag(GameTag::CHARGE) != 1)
        minion.SetExhausted(true);

    for (auto& aura : auras)
        aura->SetToBeUpdated(true);

    minion.orderOfPlay = minion.owner->GetGame()->GetNextOOP();

    ActivateAura(minion);
}
```

```
void Battlefield::ActivateAura(Minion& minion) {
    if (minion.card.power.GetTrigger().has_value())
        minion.card.power.GetTrigger().value().Activate(minion);

    if (minion.card.power.GetAura().has_value())
        minion.card.power.GetAura().value().Activate(minion);

    ...
}
```

```
void Aura::Activate(Entity& owner) {
    if (m_effects.empty())
    {
        Card card = Cards::FindCardByID(m_enchantmentID);
        m_effects = card.power.GetEnchant().value().effects;
    }

    Aura* instance = new Aura(*this, owner);

    owner.owner->GetGame()->auras.emplace_back(instance);
    owner.onGoingEffect = instance;

    instance->AddToField();
}
```

```
void Aura::AddToField() {
    switch (m_type) {
        case AuraType::ADJACENT:
        case AuraType::FIELD_EXCEPT_SOURCE:
            m_owner->owner->GetField().auras.emplace_back(this);
            break;
    }
}
```

```
void PlayMinion(Player& player, Minion* minion, Character* target, int fieldPos) {
    // Add minion to battlefield
    player.GetField().AddMinion(*minion, fieldPos);

    // Apply card mechanics tags
    for (const auto tags : minion->card.gameTags)
        minion->SetGameTag(tags.first, tags.second);

    // Process power tasks
    for (auto& powerTask : minion->card.power.GetPowerTask()) {
        if (powerTask == nullptr)
            continue;

        powerTask->SetSource(minion);
        powerTask->SetTarget(target);
        powerTask->Run(player);
    }

    player.GetGame()->ProcessDestroyAndUpdateAura();
}
```

```
void Game::ProcessDestroyAndUpdateAura() {
    UpdateAura();

    // Destroy weapons
    // Destroy minions
    // Process deathrattle tasks

    UpdateAura();
}
```

```
void Battlefield::RemoveMinion(Minion& minion) {
    RemoveAura(minion);

    if (minion.activatedTrigger != nullptr)
        minion.activatedTrigger->Remove();

    ...

    for (auto& aura : auras)
        aura->SetToBeUpdated(true);
    for (auto& aura : auras)
        aura->RemoveEntity(minion);
}
```

```
void Aura::RemoveEntity(Entity& entity) {
    if (&entity == m_owner)
        Remove();
    else
    {
        if (m_entities.empty())
            return;

        const auto iter = std::find(
            m_entities.cbegin(), m_entities.cend(), &entity);
        if (iter != m_entities.end())
            m_entities.erase(iter);
    }
}
```

트리거 구현

- 트리거는 카드에 따라 발동 시기가 다릅니다.
  - 플레이어가 턴을 시작할 때
  - 플레이어가 턴을 종료할 때
  - 영웅이나 하수인이 공격할 때
  - 영웅이나 하수인이 체력을 회복할 때
  - 영웅이나 하수인이 피해를 줄 때
  - 영웅이나 하수인이 피해를 받았을 때
  - …

트리거 구현

- 따라서 트리거를 등록할 때는 발동 시기, 조건, 효과를 함께 등록해야 합니다.

```
// ----------------------------------- MINION - PRIEST
// [CS2_235] Northshire Cleric - COST:1 [ATK:1/HP:3]
// - Set: Core, Rarity: Free
// ---------------------------------------------------
// Text: Whenever a minion is healed, draw a card.
// ---------------------------------------------------
power.ClearData();
power.AddTrigger(Trigger(TriggerType::HEAL));
power.GetTrigger().value().triggerSource = TriggerSource::ALL_MINIONS;
power.GetTrigger().value().singleTask = new DrawTask(1);
cards.emplace("CS2_235", power);
```

```
void Battlefield::AddMinion(Minion& minion, std::size_t pos) {
    m_minions.at(pos) = &minion;
    ++m_numMinion;

    if (minion.GetGameTag(GameTag::CHARGE) != 1)
        minion.SetExhausted(true);

    for (auto& aura : auras)
        aura->SetToBeUpdated(true);

    minion.orderOfPlay = minion.owner->GetGame()->GetNextOOP();

    ActivateAura(minion);
}
```

```
void Battlefield::ActivateAura(Minion& minion) {
    if (minion.card.power.GetTrigger().has_value())
        minion.card.power.GetTrigger().value().Activate(minion);

    if (minion.card.power.GetAura().has_value())
        minion.card.power.GetAura().value().Activate(minion);

    ...
}
```

```
void Trigger::Activate(Entity& source) {
    Trigger* instance = new Trigger(*this, source);
    Game* game = source.owner->GetGame();

    source.activatedTrigger = instance;

    switch (m_triggerType) {
        case TriggerType::HEAL:
            game->triggerManager.healTrigger =
                std::bind(&Trigger::Process, instance, std::placeholders::_1,
                          std::placeholders::_2);
            break;
        case TriggerType::ATTACK:
            game->triggerManager.attackTrigger =
                std::bind(&Trigger::Process, instance, std::placeholders::_1,
                          std::placeholders::_2);
            break;
    }
}
```

트리거 구현

- TriggerManager를 통해 트리거를 등록하면 발동 시기에 맞춰서 실행합니다.

```
class TriggerManager {
 public:
    void OnStartTurnTrigger(Player* player, Entity* sender);
    void OnEndTurnTrigger(Player* player, Entity* sender);
    void OnHealTrigger(Player* player, Entity* sender);
    void OnAttackTrigger(Player* player, Entity* sender);

    std::function<void(Player*, Entity*)> startTurnTrigger;
    std::function<void(Player*, Entity*)> endTurnTrigger;
    std::function<void(Player*, Entity*)> healTrigger;
    std::function<void(Player*, Entity*)> attackTrigger;
};
```

```
void Trigger::Activate(Entity& source) {
    Trigger* instance = new Trigger(*this, source);
    Game* game = source.owner->GetGame();

    source.activatedTrigger = instance;

    switch (m_triggerType) {
        case TriggerType::HEAL:
            game->triggerManager.healTrigger =
                std::bind(&Trigger::Process, instance, std::placeholders::_1,
                          std::placeholders::_2);
            break;
        case TriggerType::ATTACK:
            game->triggerManager.attackTrigger =
                std::bind(&Trigger::Process, instance, std::placeholders::_1,
                          std::placeholders::_2);
            break;
    }
}
```

```
void Character::TakeHeal(Entity& source, int heal) {
    if (GetDamage() == 0)
        return;

    int amount = GetDamage() > heal ? heal : GetDamage();
    SetDamage(GetDamage() - amount);

    owner->GetGame()->triggerManager.OnHealTrigger(nullptr, this);
    owner->GetGame()->ProcessTasks();
}
```



```
void TriggerManager::OnHealTrigger(Player* player, Entity* sender) {
    if (healTrigger != nullptr) {
        healTrigger(player, sender);
    }
}
```

```
void Trigger::ProcessInternal(Player* player, Entity* source) {
    singleTask->SetSource(m_owner);

    if (source != nullptr) {
        singleTask->SetTarget(source);
    } else {
        auto enchantment = dynamic_cast<Enchantment*>(m_owner);
        if (enchantment != nullptr && enchantment->GetTarget() != nullptr)
            singleTask->SetTarget(enchantment->GetTarget());
        else
            singleTask->SetTarget(nullptr);
    }

    if (fastExecution)
        singleTask->Run(*player);
    else
        m_owner->owner->GetGame()->taskQueue.push_back(singleTask);
}
```

정책 클래스 구현

- BasicPolicy 클래스

```
class BasicPolicy : public IPolicy {
 public:
    //! Gets next behavior from given \p game.
    TaskMeta Next(const Game& game) override;

    //! Gets proper requirement with given \p player and \p id.
    TaskMeta Require(Player& player, TaskID id) override;

    //! Notify serialized data to IPolicy.
    void Notify(const TaskMeta& meta) override;
```

정책 클래스 구현

- BasicPolicy 클래스

```
    private:
        //! Virtual method for MulliganTask requirement.
        virtual TaskMeta RequireMulligan(Player& player);
        //! Virtual method for PlayCardTask requirement.
        virtual TaskMeta RequirePlayCard(Player& player);
        //! Virtual method for AttackTask requirement.
        virtual TaskMeta RequireAttack(Player& player);
        //! Virtual method for OverDraw notifying.
        virtual void NotifyOverDraw(const TaskMeta& meta);
    };
```

정책 클래스 구현

- BasicPolicy 인터페이스
  - IPolicy의 Requirement()와 Notify()를 세분화합니다.
  - Next() 함수 : 게임에서 다음에 수행할 행동을 반환합니다.
  - RequireMulligan() 함수 : 멀리건을 수행 할 카드 정보를 반환합니다.
  - RequirePlayCard() 함수 : 카드를 내는 과정에서 필요한 대상 정보를 반환합니다.
  - RequireAttack() 함수 : 공격 주체와 공격 대상 정보를 반환합니다.
  - NotifyOverDraw() 함수 : 핸드에 카드가 10장이 넘어 탔다는 정보를 처리합니다.

정책 클래스 구현 예시

- RandomPolicy 클래스

```
class RandomPolicy : public BasicPolicy {
 public:
    //! Gets next behavior with random actions.
    TaskMeta Next(const Game& game) override;
 private:
    //! Method for MulliganTask requirement.
    virtual TaskMeta RequireMulligan(Player& player);
    //! Method for PlayCardTask requirement.
    virtual TaskMeta RequirePlayCard(Player& player);
    //! Method for AttackTask requirement.
    virtual TaskMeta RequireAttack(Player& player);
};
```