<PROJECT 중간보고서>

Two of Scorched Earth

탈 중앙 검색엔진 서비스의 이해 및 증명을 위한 웹 게임 개발

2014-11217 공은채

2018-16371 문보설

2014-11476 차우진

2019-10423 Charles Onyango

엄현상 교수님

임원섭 담당자님 (Ethereum Foundation, Applied ZKP developer)

Table of Contents

1. Abstract

이 프로젝트에서는 탈 중앙 검색엔진 서비스를 이해시키고 홍보하기 위한 게임 형태의 introduction 을 만드는 것을 목적으로 한다. 이 게임의 핵심 기능은 인트로 모듈, 역할 체험 모듈, 샌드박스 모듈 그리고 크레딧 모듈이다. 이 게임이 목표로 하는 바는 탈 중앙 검색엔진 서비스의 핵심 아이디어를 사용자들에게 이해시키는 것과 해당 서비스를 자신의 서비스에 도입하고자 하는 개발자들에게 도움을 주는 것이다. 또한 오픈소스 접근성을 높이고 실제 응용 사례를 보여주는 것도 이 게임의 목표라고 할 수 있다. 이를 위해 게임을 인트로 모듈, 역할 체험 모듈, 샌드박스 모듈, 크레딧 모듈로 구성한다. 인트로 모듈에서 게임 사용자들은 탈 중앙 검색엔진 서비스의 등장배경과 문제의식, 간단한 개념 소개를 받으면서 서비스의 핵심 아이디어를 이해할 수 있다. 그리고 이어서 나오는 역할 체험 모듈에서 게임 사용자들은 탈 중앙 검색 엔진 서비스 내의 컨텐츠 제공자와 컨텐츠 구매자의 역할을 각각 경험하면서, 서비스 매커니즘을 이해할 수 있다. 다음으로 샌드박스 모듈에서 게임 사용자들은 컨텐츠 거래에 영향을 미치는 factor 들을 조정해 가며 거래 결과값을 확인할 수 있으며, 이를 통해 핵심 아이디어를 심화적으로 이해할 수 있다. 또한, 소프트웨어 개발자는 이를 통해 탈 중앙 검색엔진 오픈소스를 자신의 서비스에 적용할지 고려할 수 있을 것이다. 마지막으로 크레딧 모듈에서 게임 사용자들은 이더리움이 현재 개발 중인 탈 중앙 검색엔진 오픈소스의 링크와, 이 오픈소스가 도입된 사례를 볼 수 있다. 이를 통해 오픈소스의 낮은 접근성과 실제 응용 사례의 부재를 보완할 수 있다.

2. Introduction

탈 중앙 검색엔진 서비스란 제 3자가 거래를 중개하는 기존 검색엔진 패러다임에서 탈피하여, 검색 정보 제공자와 검색 정보 구매자의 직접 거래를 제공하고자 하는 서비스이다. 이더리움 재단에서는 탈 중앙 검색엔진 서비스를 개발하면서, 개인 간 거래의 낮은 거래 신용도라는 단점을 극복하기 위해 Two of Two Scorched Earth라는 개념을 도입했다. 이 개념은 전자 상거래에 구매자의 보복 가능성을 추가하여 구매자가 서비스에 불만족 했을 경우 제안자의 deposit을 태울 수 있도록 한다. 이를 통해 제안자는 좀 더 책임감 있고 높은 질의 서비스를 제공할 수 있게 된다.

이더리움에서 개발 중인 해당 서비스는 상당히 참신하고, 성공한다면 모든 전자상거래의 패러다임을 바꿀 수 있는 서비스라고 생각된다. 또한 거래 중개자의 권력 및 수수료를 거래

당사자가 나누어 가짐으로써 더욱 효율적인 거래를 가능하게 할 수 있다. 그러나 우리는 이 서비스 오픈소스의 현실적 파급력 문제와 서비스 자체의 합리성 문제에 있어 몇 가지 Pain Point 들을 짚어냈고, 이것들은 서비스 자체의 영향력에 치명적인 지점들이라고 판단했다. 따라서 이를 해결하기 위해 서비스에 대한 introduction 을 제공하는 게임을 만들고자 한다.

첫 번째 Pain Point 는 탈 중앙 검색엔진 서비스의 핵심 아이디어를 이해하기 어렵다는 것이다. 탈 중앙 검색엔진 서비스는 우리의 일상생활에 부합하는 중개자 기반 서비스를 정면으로 반박하므로, 탈 중앙 패러다임 및 블록체인에 대한 이해가 없는 사람이라면 이해하기 어렵다. 또한 여기 사용된 two of two scorched earth 개념 역시 행동학적 분노 모델과 게임 이론에 대한 이해가 부족하다면 이해하기 어렵다. 오픈소스의 스펙 문서가 해당 서비스가 제공하는 유일한 설명인데 이는 상당히 기술적 관점에서 쓰여 있어 기술을 알지 못하는 사람에게는 큰 도움이 되지 못한다. 이러한 점들 때문에 서비스를 이해하는 사람들이 적을 것으로 예상되고, 이로 인한 파급력의 저하가 우려된다. 우리의 프로젝트는 이 문제점을 해결하는 것을 첫 번째 목표로 한다. 우선 우리의 프로젝트는 인트로 모듈에서 게임 설명의 형태로 탈 중앙 검색엔진 오픈소스의 등장 배경과 문제 의식, 핵심 개념을 보다 쉽게 소개한다. 또한, 거래에 등장하는 각 역할을 체험할 수 있는 역할 체험 모듈과 전지적으로 거래에 영향을 미칠 factor 들을 조정하고 그 결과를 확인하는 샌드박스 모듈도 제공한다. 이러한 기능들을 통해 게임 사용자들은 탈 중앙 검색엔진 서비스의 기본 매커니즘과 핵심 아이디어를 쉽게 이해할 수 있다. 특히 샌드박스 모듈은 탈 중앙 검색엔진 서비스를 자신의 프로그램에 도입하고자 하는 개발자들을 도울 수도 있을 것이다.

두 번째 Pain Point 는 오픈소스의 접근성이 낮다는 것과 실제 응용 사례를 찾아보기 힘들다는 것이다. 오픈소스는 어디에도 홍보되지 않고, 깃허브에 찾아 들어가지 않는 이상 존재를 알 수 없다. 또한 해당 오픈소스에 실제 응용 사례가 언급되어 있지 않다. 이런 단점들은 이를 활용하려는 개발자들이 어려움을 겪게 하며 파급력의 저하로 이어진다. 이를 해결하는 것이 우리 프로젝트의 두 번째 목표이다. 우리는 게임의 마지막에 크레딧 모듈을 넣어, 오픈소스 링크와 오픈소스가 적용된 사례를 소개하고자 한다. 이러한 기능을 통해 오픈소스의 접근성과 활용가능성을 높일 수 있을 것이다.

세 번째 Pain Point 는 탈 중앙 검색엔진 서비스의 기본 가정 자체가 합리적이지 않은 것처럼 보인다는 것이다. 탈 중앙 검색엔진 서비스는 기본적으로 행위자의 보복 심리를 전제하고 있다. 그런데 이러한 행위자 분노 모델은 그 사실성이 의문스럽고, 경제학의 합리적 행위자 가정에 어긋나는 것이라 여겨진다. 행위자 분노 모델과 합리적 행위자 가정이 모두 성립하는 경우는 첫째로 거래가 이루어지는 재화가 구매자에게 필수불가결한 것이며, 거래 상황에서

제안자의 수가 극도로 적은 경우라고 생각된다. 그러나 이런 거래 경우는 경험적으로 많지 않으며, 따라서 오픈소스는 제한된 적용범위를 가지는 것이다. 혹은 많은 수의 구매자가 경제적으로 합리적인 선택보다, 그들이 생각하는 정의로운 선택을 하는 경우에도 해당 오픈소스가 적용될 수 있다. 그러나 이런 경우 역시 경험적으로 많지 않은 것 같다. 즉 어느 경우에든 탈 중앙 검색엔진 서비스는 제한된 적용 범위를 가지므로, 해당 서비스를 다른 프로그램에 도입하기 위해서는 이 부분을 충분히 고려해야 한다. 이 단점은 단지 오픈소스의 적용범위를 좁히는 것이 아니라, 오픈소스가 무분별하게 적용되었을 때 서비스의 패러다임 자체에 대한 의문을 초래할 수 있는 심각한 단점이다. 따라서 우리의 게임은 이러한 위험성을 인지하고, 해당 서비스를 도입하고자 하는 개발자들이 충분히 이 부분을 고려할 수 있도록 하는 것을 마지막 목표로 한다. 이를 위해서 샌드박스 모드에서 제안자 수와 구매자 수를 factor 로서 조정할 수 있게 할 것이다. 차후에 서비스가 필수불가결한 정도 그리고 정의감이 높은지 등의 factor 를 추가하는 것도 고려 중에 있다. 이를 통해 해당 오픈소스를 자신의 프로그램에 도입하고자 하는 사람들은 각 요소가 거래의 성공과 실패에 큰 영향을 미친다는 점을 알 수 있고, 이 부분에 대한 경각심을 가지고 도입을 숙고할 수 있다.

3. Background Study

A. 관련 접근방법/기술 장단점 분석

이번 프로젝트에서는 페이스북에서 제공해주는 프론트엔드 라이브러리인 React.js 를 활용할 예정이다. 구현할 게임이 가볍고 현대적이어야 한다는 점에서 Angular 보다는 React 를 선택하는 것이 적절하다고 판단하였기 때문이다. 또한 많은 이더리움 프로젝트가 React 기반의 개발을 하고 있다는 점, React 가 가장 기술 스택이 많이 쌓여 있다는 점 등도 React 를 선택한 이유 중 하나이다.

또한 React 를 활용하면 컴포넌트를 활용할 수 있기 때문에, 우리 게임의 다양한 UI 디자인 구현 및 유지 보수성 측면에서 유리할 것이라 판단했다. React 에서는 컴포넌트를 통해 UI 수정 및 재사용이 간단히 해결할 수 있다. 이를 통해 코드 재 사용성 및 유지보수성을 증가시킬 수 있다.

서버의 경우 게임 특성상 데이터 축적이 필요한 기능이 없기 때문에 static server 호스팅을 사용하기로 결정했다. 이를 구현하기 위해서는 node 의 serve, AWS 나 Azure 가 제공하는 클라우드 컴퓨팅 서버 등을 활용할 예정이다.

위 내용들에 대한 구체적인 내용은 6.B Architecture Description 에 기술되어 있다.

B. 프로젝트 개발환경

4 명으로 구성된 팀이며, 이더리움 재단과의 협업이 매우 중요하다고 판단하여 버젼 관리 도구로 git 을 활용해 각각 개발을 진행할 것이다. 원격 저장소로는 github 을 사용하고, issue 및 wiki 기능을 활용해 협업의 효율성을 증가시킬 것이다. 또한, VSCode 를 활용하여 효율적인 개발을 진행할 것이다.

웹 게임 개발을 진행하는 과정에 있어, 보편적으로 활용되는 html 과 css, javascript 를 활용할 것이다. 또한, Background Study A 에서 언급한 바와 같이 페이스북에서 제공해주는 프론트엔드 라이브러리인 React.js 를 활용할 예정이다.

또한, 비주얼 디자인 부분은 피그마를 통해서 wireframe 등 전체적인 디자인 작업을 진행하고, 정교한 이미지가 들어가는 경우에는 어도비 포토샵과 어도비 일러스트레이터를 통해서 구현할 예정이다.

4. Goal/Problem & Requirements

이번 프로젝트의 목표는 탈 중앙 검색엔진 서비스의 등장 배경과 문제의식, 핵심 아이디어에 대한 이해를 증진하고 접근성을 높이며, 개발자들에게는 오픈소스 도입 고려를 도울 수 있는 게임을 제작하는 것이다. 기본적인 매커니즘 및 핵심 아이디어에 대한 이해를 돕는 것 뿐만 아니라, 거래에 영향을 주는 factor 와 해당 서비스의 적용 가능성을 검토할 수 있도록 하는 것을 목적으로 한다.

이번 프로젝트가 성공하기 위해서는 우선 탈 중앙 검색엔진 서비스의 기본 매커니즘과 핵심 아이디어를 게임을 통해 쉽게 전달할 수 있어야 한다. 특히, UI 디자인 측면에서 사용자가 거부감 없이 쉽게 접근 가능해야 한다. 그리고 각 화면의 설명 역시 쉽고 요약적으로 쓰여져야 한다. 또한, 여러 factor 를 도입하는 데 있어 충분한 논의가 필요하다. 그리고 이러한 노력으로 전달된 오픈소스 아이디어가 정확한 것이어야 하므로, 기존 오픈소스에 대한 정확한 이해 역시 필요하다.

5. Approach

앞서 말했듯이 이번 프로젝트 수행 목표는 탈 중앙 검색엔진 서비스에 대한 introduction 을 제공하는 게임을 만드는 것이다. 이를 통해 탈 중앙 검색엔진 서비스에 대한 홍보 및 이해를 증진하고 서비스 도입에 대한 고려를 돕는 것이 궁극적인 목표라고 할 수 있다. 이러한 목표를 달성하기 위해 앞서 설명한 인트로 모듈, 역할 체험 모듈, 샌드박스 모듈, 크레딧 모듈 네 가지 기능을 게임 내에서 제공하고자 한다. 이 네 가지 모듈을 구현하기 위해서는 다음과 같은 사항들이 필요하다.

우선 역할 체험 모듈에서는 게임 사용자가 역할 선택이나 행동 선택에서 클릭하는 버튼들에 따라 적절한 페이지를 보여줘야 한다. 이 부분은 React event handler 를 사용하여, 사용자가 화면에서 어떤 버튼을 클릭하였는지를 Input 으로 받아 적절한 페이지로 라우팅한다. 또한 샌드박스 모듈에서는 게임 사용자의 Input 을 받아 시뮬레이션을 수행하고, 그 결과값을 산출하는 시뮬레이션 코드가 필요하다. 이 결과값으로 모든 agent 의 거래 결과를 전달할 수도 있고 통계치만을 전달할 수도 있는데, 우리 프로젝트에서는 통계치를 전달하도록 한다. 앞서 언급했듯 샌드박스 모듈의 목적 중 하나는 서비스 개발자의 고려를 돕는 것이고, 서비스 개발자에게는 통계치가 더 필요한 정보이기 때문이다. 또한 너무 많은 정보를 줄 경우 이해가 어려울 수 있어, 이해를 돕는다는 목표를 달성하지 못할 수 있다. 또한 샌드박스 모듈은 그 역할에 따라 시뮬레이션을 수행하는 부분과 화면 출력 및 사용자 입력을 관리하는 부분으로 나누어 구성하고자 한다.

프로젝트 구현에서 백엔드 API 없이 프론트엔드에 React.js 를 사용한다. 이 선택의 주된 이유는 게임 시뮬레이션에 사용자 인증과 같은 백엔드 서비스나 데이터베이스가 필요하지 않기 때문이다. 따라서 우리는 앞서 언급한 프레임워크를 사용하여 SPA(단일 페이지 애플리케이션)를 만드는 모델을 사용하기로 결정했다.

게임 UI 디자인 또한 중요하다. 탈 중앙 검색엔진 서비스와 관련된 사용자 경험을 개선하기 위해서는, 직관적이고 깔끔하면서도 유저 입장에서 심플하게 내용을 받아들일 수 있는 디자인이 중요한 요소라고 판단했기 때문이다. 이더리움 재단의 디자인 방법론[1]과 브랜드 가치에

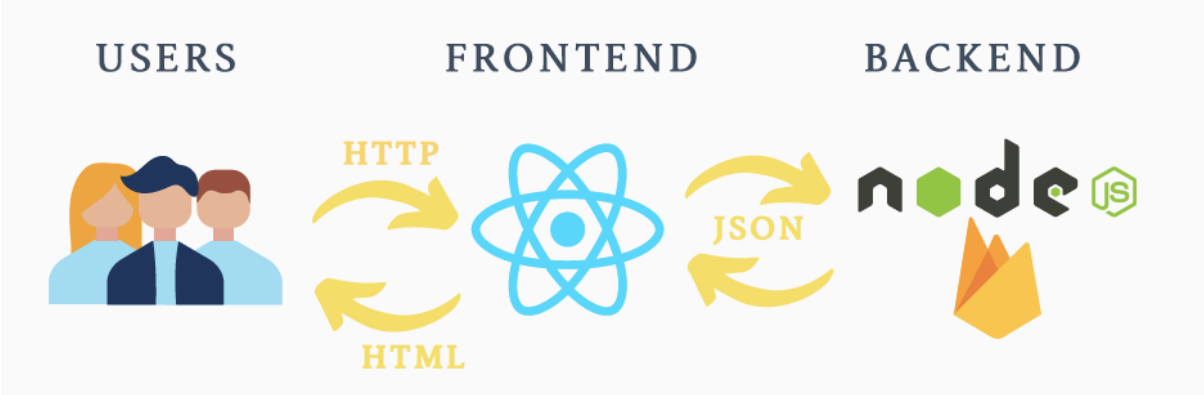
[1] "Design principles | ethereum.org", Ethereum homepage, last modified Sep 6, 2021, last accessed

입각한 UX 측면에서 효율적인 UI 디자인을 진행하기 위하여, Figma 를 활용하여 전체적인 시각적 인터페이스 디자인을 진행하고 있다. 향후 Adobe Creative Suite 프로그램 중 Photoshop 과 Illustrator 를 활용하여 게임에 활용할 캐릭터를 기업 담당자와의 회의를 통해 협의하려 한다.

마지막으로 최종 결과를 시연할 때는 완성된 게임을 실시간으로 실행하는 모습을 보여주고자 한다. 이 과정이 길어질 경우 동영상을 녹화한 뒤, 이를 압축된 형태로 편집하여 결과물을 제출하려고 한다. 또한 이더리움 재단 측의 도움을 받아 실제 시뮬레이션 결과를 받고, 이것이 우리 게임의 결과와 같은지 비교하여 서비스를 검증할 예정이다.

6. Project Architecture

A. Architecture Diagram



B. Architecture Description

프론트엔드에서는 페이스북에서 제공해주는 프론트엔드 라이브러리인 React.js 를 활용할 예정이다. Vue 와 Angular 등 다양한 웹 프레임워크/라이브러리가 아닌 React.js 를 선택한 이유는 다음과 같다. 이 게임 프로젝트는 짧은 기간 안에 가벼우면서도 현대적인 애플리케이션을 개발해야 하기 때문에, 기능이 다양하고 규모가 큰 애플리케이션을 만들 때 유리한 Angular 보다는 React 를 선택했다. Vue 도 작고 가벼운 애플리케이션을 빠르게 개발하기에 유리한 면이 많지만, Ethereum Foundation 의 Scorched Earth 프로젝트가 React 를 기반으로 개발이 되었다는

Oct.27, 2021, https://ethereum.org/en/contributing/design-principles/

점과 2021 년 현재 웹 개발 커뮤니티에서 가장 많이 활용되는 javascript library 라는 React 의 특성 때문에 React 를 선택하게 되었다.

그리고 다양한 UI 디자인 요소가 해당 웹 게임 프로젝트에 들어가기 때문에, React.js 를 활용하면 컴포넌트를 활용하여 간편하게 UI 를 수정하고 재사용이 가능할 것이라고 판단을 하였다. React.js 의 컴포넌트를 활용하면 헤더, 메인 콘텐츠, 버튼, 사이드바 메뉴, 캐릭터 등을 헤더 컴포넌트, 사이드바 컴포넌트와 같이 하나의 컴포넌트로 묶어서 관리할 수 있다. 그렇기 때문에, 컴포넌트를 활용하여 코드의 재사용성과 유지보수성을 증가시켜 줄 계획이다. 이러한 특성들 때문에, React 를 사용하면 효율적으로 프로젝트 개발 및 관리를 진행할 수 있다고 판단하였다.

또한 서버 부분은 해당 프로젝트에서 smart contract 를 연결 하지 않고, 시뮬레이션을 프론트로 이동하기 때문에 static server 와 aws 혹은 azure 등으로 간단히 구현하기로 한다. 구체적 구현 방법을 논하기 전에, Smart contract 를 연결하지 않은 이유와 시뮬레이션을 프론트에 위치시킨 이유를 설명하고자 한다.

Smart contract 개발의 경우 탈 중앙 검색엔진 서비스 구현에 필요한 부분이지만, 우리 프로젝트는 탈 중앙 검색엔진 서비스의 기본 아이디어를 이해시키기 위해 시뮬레이션 게임을 구현하는 데 중점을 둔다. 왜냐하면 서비스를 구현하는 부분과 서비스의 pain point 를 잡는 것 중 하나를 선택해야 하는 상황에서, 후자가 더 중요성을 가진다고 판단했기 때문이다. 먼저 smart contract 를 구현 및 연결하기 위해서는 solidity 언어 및 ether.js, typescript, 그리고 hardhat 이나 nitro-protocol, openzepplin 등의 라이브러리를 깊게 공부해야 한다. 그런데 이 부분은 웹 게임 개발과 병행하기 시간적으로 무리가 있다. 또한 우리가 발견한 Pain point 들은 서비스 구현과는 독립된 속성들(접근성, 이해 난이도)이거나 서비스 패러다임 자체를 바꾸어야 하는 속성들(아이디어 합리성)이기 때문에, 서비스 구현과 병행하여 해결하기 어렵다. 즉 서비스의 pain point 를 보완하는 것과 서비스를 실제로 구현하는 것 중에서 하나를 선택해야 하는 상황이라고 할 수 있다. 그리고 이러한 상황에서 우리는 전자가 우선시된다고 판단했다. 앞서 말했듯 pain point 들이 서비스 파급력에 치명적인 영향을 가질 수 있어서, 이러한 pain point 를 먼저 해결하지 못한다면 서비스의 목적이 현실화되지 않을 수 있기 때문이다. 이에 관련하여 기업 담당자님과 이야기를 나눈 결과, 담당자님께서는 이번 학기에는 pain point 보완 및 웹 시뮬레이션 게임에 집중하는 접근법을 추천하셨다. 따라서 이번 프로젝트에서는 smart contract 구현 보다는 웹 시뮬레이션 게임 개발에 더욱 집중하는 것을 목표로 하고자 한다.

또한 이번 프로젝트에서 샌드박스 모듈를 구현하기 위해 필요한 시뮬레이션 코드는 데이터 축적을 필요로 하지 않으므로 프론트에서 구현하기로 결정했다. 우선 샌드박스 모듈에서의 결과 화면은 매 시행의 독립적인 결과치를 결과 화면에 보여주므로, 이전 시행에 대한 정보를 필요로 하지 않는다. 이러한 특성 때문에 시뮬레이션 코드는 프론트에서 돌아가며 결과값을 관리하는 것이 더 적절하다고 판단하였다.

위의 이유들로 smart contract 연결을 하지 않고, 시뮬레이션을 프론트로 이동하였기 때문에 서버의 역할이 축소되게 되었다. 또한 게임의 특성상 회원 관리나 데이터 축적이 필요하지 않다는 점에서도 서버의 중요성이 떨어진다고 생각되어, 이 프로젝트에서는 static server 호스팅을 사용할 계획이다. 이를 위해서 우선 Node 의 serve 를 사용하여 npm run build 를 실행한다. 그리고 그 build 결과를 서버를 통해 배포한다. 배포 과정에서는 Amazon AWS 또는 Microsoft Azure 와 같은 클라우드 컴퓨팅 서비스를 제공하는 서버를 활용할 예정이다. 사용할 수 있는 대체 솔루션에는 Git Pages 또는 Netlify 또는 Firebase 와 같은 서버리스 클라우드 호스팅 솔루션이 있다. React 앱을 위한 production build 를 생성하는 과정은 'npm run build'를 통해 간단히 수행되며 결과 build 폴더는 앞서 언급한 솔루션 중 하나를 통해 배포 및 호스팅할 수 있다.

7. Game Flow

A. Game Wireframe

*별도 첨부한 pdf 파일에 동일한 그림이 있음

B. Game Flow Description

Wireframe 에 그려진 게임 흐름에 대해 페이지별로 어떠한 정보 및 요소가 들어가는지 간단히 서술하고자 한다.

게임의 여러 페이지들은 총 25 개의 React View 간의 전환으로 구성되어있으며 가장 먼저 Start 페이지에서 출발한다. Start 페이지에서는 게임의 제목 및 게임 진행에 소요되는 시간, 창작자들의 이름 관련 정보를 제공한다.

게임의 목적 중 하나는 탈 중앙 검색엔진 서비스의 핵심 개념을 전달하는 것이기 때문에 이후에는 프로젝트의 출발점, 배경, 개념에 대한 서술 페이지인 Problems, Background, ProjectConcept 페이지가 뒤따른다. 각 페이지에는 서술하는 내용에 관한 텍스트 필드와 이를 직관적으로 전달하는 이미지 파일이 각각 포함되어 있다.

이후 게임 사용자는 Scorched Earth 프로젝트의 두 당사자인 Suggester 와 User 중 한 역할을 선택한다. 선택한 역할에 따라 Suggester 와 관련한 페이지들이 나열되거나 User 와 관련된 페이지들이 나열된다. 게임 사용자는 자신이 선택한 역할 체험이 끝난 뒤에 다시 역할 선택 페이지로 로드되어 남은 역할을 체험해보거나 Withdraw 버튼을 클릭하여 사용자들이 거래를 중지하는 방안에 대해 알아볼 수도 있다. 페이지들의 각 요소에 마우스가 hover 하였을 때 ReactDOM 의 onMouseOver 로 이벤트를 받아와 해당 개념과 관련된 설명을 볼 수 있도록 처리하여 게임 사용자들의 이해를 증진시키려한다. 역할 체험과 관련된 모든 페이지 하단에는 공통적으로 Footer 컴포넌트가 들어가 게임 백그라운드 사운드를 끄거나 켤수 있는 토글버튼과 게임의 특정 페이지로 넘어갈 수 있는 리다이렉트 기능을 가진 버튼 리스트가 제공된다.

샌드박스 페이지에서 게임 사용자는 factor 를 조정한다. factor 는 suggester 의 수, suggester 의 rate, suggester service 수준, user 의 수, user 의 인내심, deposit, suggester 가 태워야 하는 양, user 가 태워야 하는 양, user 가 지불해야 하는 양이다. 해당 factor 들을 조정하고 나서 Game Start 버튼을 누르면 시뮬레이션이 시작된다. 시뮬레이션 진행 도중에는 suggester 와 user 가 거래를 하면서 변화되는 값들을 볼 수 있다. 이 값은 각 agent 의 deposit 이다. 게임 사용자가 stop 버튼을 누르면 결과를 보여주는 화면으로 넘어간다. 결과 화면에서는 User 의 평균 처벌횟수, 평균 보상횟수, suggester 의 deposit 이다. suggester 의 service 수준 증가율은 deposit 이 거래 성공 혹은 실패를 보장하지 않는다고 생각될 경우 추가할 가능성이 있다.

샌드박스 페이지 이후에는 해당 프로젝트에 관련해 더 생각해볼 지점들이 무엇인지에 대해 전달하는 limitation 페이지와 해당 프로젝트와 연관된 다른 프로젝트들이 나열된 크레딧 모듈 화면이 순차적으로 배치된다.

8. Implementation Spec

A. Input/Output Interface

1. Input

이번 프로젝트에서는 게임사용자와 웹페이지 사이의 상호작용 과정에서 대부분의 input 이 발생할 것이다. 웹페이지에서 input 값을 주는 방법은 다양하지만, 사용자의 편의성을 증대시키기 위해 프로젝트에 대해 설명하는 페이지에서 대부분의 input 은 각 요소에 대한 클릭값으로 설정하였다.

프로젝트에 관한 아이디어, 배경 및 개념에 대해 소개하는 페이지에서는 <nextButton /> 컴포넌트의 state 가 항상 isEnabled: true 이므로 단순히 해당 컴포넌트를 클릭하는 것만으로 다음 페이지로 넘어갈 수 있다. 이 때 link 를 통해 지정한 url 로 이동하게되는데, 이는 아예 새로운 페이지를 불러오게 되므로 기존 컴포넌트 상태값은 소멸된다. 이는 한계 및 크레딧 페이지에서도 동일하게 적용된다.

역할체험 페이지에서 사용자가 각 페이지에서 요구되는 사항들을 모두 만족시키면 다음 페이지로 넘어갈 수 있는 버튼을 클릭할 수 있게 <nextButton /> 컴포넌트의 state 가 isEnabled: true 로 바뀐다. <nextButton />이 활성화된 상태에서 사용자가 해당 버튼을 클릭하면 React Router 를 통해 다음 페이지로 라우팅된다. 각 페이지마다 요구하는 사항들은 차이가 있으나, React DOM 에서 제공하는 사용자 친화적인 이벤트 리스너(마우스 이벤트 리스너, 이미지 이벤트 리스너, 트랜지션 이벤트 리스너, 키보드 이벤트 리스너)를 주로 활용하여 게임 조작성을 증대시키고자 한다. 역할체험 페이지 하단의 <footer /> 컴포넌트는 역할체험 페이지들 중 원하는 페이지로 바로 이동할 수 있는 기능을 제공하며 이 때도 사용자의 onClick 이벤트를 받아 Router 가 동작하게 된다.

위 과정을 통해 게임 사용자가 프로젝트의 개념과 Suggester, User 의 상호작용 과정에 대해 이해했다면, 그 후 사용자는 샌드박스 모드를 통해 프로젝트의 가설이 실제로 성립하는지를 확인하기 위해 시뮬레이션 코드에 파라미터로 전달될 10 가지의 input 값들을 조정하게 된다. 이 때의 input 값은 크게 거래상황과 관련된 input 들과 처벌/보상에 관련된 input 들로 구분할 수 있다.

- 거래 상황 관련 inputs: Suggester 의 수, Suggester 의 rate, User 의 수, User 의 인내심, Deposit
- 처벌/보상 관련 inputs: Suggester 가 태워야하는 양, User 가 태워야하는 양, User 가 지불해야하는 양

이 때 수치화 할 수 있는 input 값은 user 가 React TextField 에 입력하는 값으로 양의 정수 및 실수값으로 규정되어 있어, 그 외의 값이 들어올 경우 alert 를 통해 user 에게 올바른 input 값을 입력하기를 요구한다. 후에 추가할 가능성이 있는 정의감과 같은 정성적 요소는 상/중/하의 세 단계로 나누어 게임 사용자가 가장 최근에 클릭한 한 버튼값만 저장하여 시뮬레이션 코드에 전달할 수 있다.

2. Output

이번 프로젝트의 output 은 대부분 컴포넌트 내부에서 호출된 render() 함수의 결과 생성되는 html 로 웹 게임의 화면을 그리는데 사용된다. render() 함수는 DOM 에 html 을 내보낼 때 호출될 수도 있고 컴포넌트의 state 가 바뀌어 setState() 함수가 실행되었을 때 자동으로 호출되어 화면을 다시 그릴수도 있다.

샌드박스 모듈에서는 시뮬레이션 코드에서 나온 output 값들을 기반으로 render() 함수를 호출한다. 이 때 시뮬레이션 코드에서 나온 output 은 suggester 의 deposit 과 user 의 총 거래 횟수, 보상 횟수, 처벌 횟수이다.

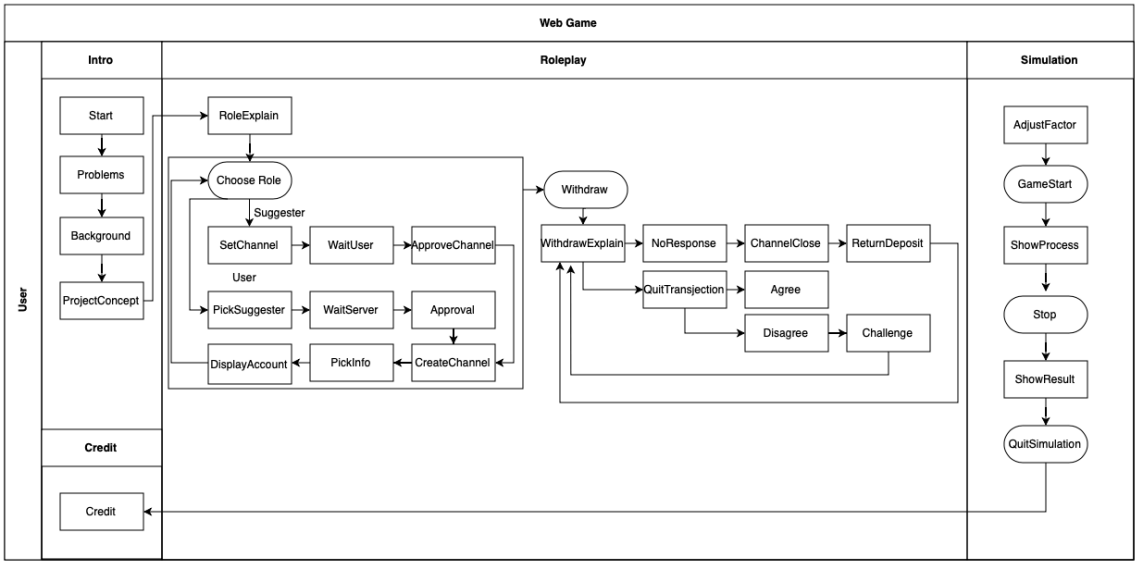
B. Inter Module Communication Interface

게임 내에서 communication이 일어나는 module은 두 가지 분류를 가진다. 첫 번째로 앞서 설명한 네 가지 기능들(인트로 모듈, 역할 체험 모듈, 샌드박스 모듈, 크레딧 모듈)을 각각의 모듈로 보고, 그러한 모듈들 사이의 전환이 어떻게 이루어지는지를 살펴볼 수 있다. 두 번째로 샌드박스 모듈 내에서 시뮬레이션이 실제로 수행되는 코드 부분과 사용자로부터 입력 값을 받아 시뮬레이션에 전달하고 시뮬레이션 후 산출된 결과를 받아 화면을 그리는 부분을 구분할 수 있다. 전자를 시뮬레이션 모듈이라고 부르고, 후자를 IO모듈이라고 부르면, 샌드박스 모듈은 이 두가지 모듈로 구성되어 있는 것이다.

첫 번째 분류에 나타난 모듈 간의 상호작용은 상호 모듈의 전환이라고 할 수 있다. 네 가지 모듈들의 전환은 사용자가 다음 모듈이 시작되는 페이지로 가는 버튼을 클릭했을 때이다. 네 모듈이 차례로 등장할 것이므로, 각 모듈의 마지막 페이지에서 다음 페이지로 진행하는 버튼 혹은 해당 모듈이 제공하는 기능을 종료하는 버튼을 눌렀을 때 전환이 일어나는 것이다. 이러한 화면 전환은 nextPage()함수를 통해 이루어진다. nextPage()함수는 React router와 react의 event handler를 이용하여 구현된다. 이 함수에서는 event handler를 활용하여 다음 페이지 버튼을 눌렀을 때 next를 호출하는데, next는 react-router-dom의 history를 이용하여 다음 페이지로 라우팅한다. 구체적 함수 내용은 Appendix의 nextPage()함수를 참고하면 된다.

두 번째 분류에 나타난 시뮬레이션 모듈과 IO 모듈의 상호작용은 IO모듈에서 시뮬레이션 모듈을 호출하는 것일 것이다. 이것은 react의 useEffect를 이용하여 만든 커스텀 Hook을 통해 일어난다. useEffect를 이용하여 importSimulation(res)이라는 함수를 구현하고, 이 함수를 통해 컴포넌트에 res를 path로 하는 외부 자바스크립트 파일을 컴포넌트에서 사용할 수 있도록 한다.

C. Modules



탈 중앙 검색엔진 서비스를 위한 웹게임 개발 프로젝트는 4 가지의 모듈로 구성되어 있다. 이번 프로젝트에서는 유저가 게임을 플레이하는 흐름에 따라서 유저의 활동을 4 가지로 분류하고, 이를 각각 하나의 모듈로서 작동하게 설계하였다.

첫 번째는 간단한 게임 설명과 함께 현재 문제점을 지적하며 프로젝트의 배경을 소개하는 인트로 모듈, 두 번째는 User, Suggester, Withdraw 세 가지의 파트로 구성된 역할 체험 모듈이다. 그리고 세 번째는 실제로 시뮬레이션을 돌려볼 수 있는 샌드박스 모듈, 네 번째는 프로젝트의 한계를 언급하며 엔딩 크레딧 페이지를 볼 수 있는 크레딧 모듈이다.

인트로 모듈에서는 유저가 게임을 시작하면 게임에 대해서 간단하게 소개하는 화면이 등장한 다음, 현재 문제점을 지적하는 화면이 등장하고, 프로젝트 배경을 소개하는 화면이 등장한다. 그 다음에 프로젝트의 개념을 소개한 다음 역할 체험 모듈로 넘어간다.

역할 체험 모듈은 크게 User 또는 Suggester 로서 역할 체험을 할 수 있는 부분과 Withdraw 버튼을 통해서 Withdraw 를 할 수 있게 하는 부분으로 구성된다. 다음 문단에서 역할 체험을 할 수 있는 부분을 User 와 Suggester 두 가지로 나눠서 설명을 하고, Withdraw 를 할 수 있는 부분에 대해서 설명을 진행할 것이다.

Suggester 로 역할을 선택하면, 일련의 과정을 통해서 채널이 세팅된다. 그 다음, 형성된 채널에 유저가 올 때까지 기다린 다음, 채널이 승인되는 것을 기다린다. User 로 역할을 선택하면, Suggester 를 직접 선택한 다음에, 서버의 응답을 기다린 다음, 승인이 나게 된다. 그 다음, Suggester 를 선택했을 때와 User 를 선택했을 때의 시나리오는 동일하다. 채널이 형성된 다음, 정보를 고르고, 거래 후의 계정 정보를 시각화는 과정을 통해서 역할 체험 모듈이 진행된다. Withdraw 버튼을 누르게 되면, 상대방의 답장이 없을 때와 둘 중 한명이 거래에 그만 참여하고 싶은 경우 두가지로 나눠져 게임이 진행된다.

샌드박스 모듈에서는 사용자는 거래에 미치는 여러가지 factor 를 조정하면서 거래 결과를 확인할 수 있다. 앞선 역할 체험 모듈이 각각의 역할을 체험하면서 오픈소스의 기본적인 매커니즘을 이해하는 것을 그 목적으로 하고 있다면, 샌드박스 모듈은 각각의 역할의 factor 를 조정하여 전지적 시점에서 오픈소스에 대한 이해를 심화하는 것을 목적으로 한다. 인터랙티브하게 진행되는 게임을 통해서 게임 사용자들에게 오픈소스에 대한 사용자 경험 개선이 크게 일어날 것이라고 예측된다.

크레딧 모듈에서는 프로젝트의 한계를 언급하며 이 게임과 관련된 다양한 정보를 확인할 수 있다. 게임 소스코드 github 페이지를 오픈소스로 올리는 과정을 통하여 이더리움 재단의 브랜드 가치를 확장시키며, 기존 서비스 오픈소스의 github 링크를 올려서 탈 중앙 검색엔진 서비스에 대한 홍보 효과도 줄 예정이다. 또한 관련 프로젝트에 참여한 사람들의 github 링크를 같이 소개함으로서, 서비스의 확장성을 높일 것이다.

9. Current Status

현재까지 프로젝트가 진행된 사항은 다음과 같다. 경제학 논문 등을 참고하며 이 프로젝트의 합리성에 대한 논의를 진행하였고, 프로젝트의 가정에 따라 가설을 진행하는 방식으로 프로젝트를 진행하기로 결정하였다. 탈 중앙 검색엔진 서비스를 도입하려는 개발자들이 합리성이나 적절성에 대한 고려를 해 볼 수 있게끔, 시뮬레이션 factor 조정을 하는 방식으로 샌드박스 모드에서 필요한 시뮬레이션 코드를 구현하였다.

또한, 기업 담당자와의 미팅을 통하여 웹 게임을 통한 탈 중앙 검색엔진 서비스의 사용자 경험 및 파급력을 개선하기 위하여 웹 게임의 완성도를 높이는 것에 집중하기로 결정하였고, 게임 시나리오 기획과 UX 및 UI 디자인을 주기적으로 진행하고 발전시키고 있다. 전체적인 레이아웃 방향은 Figma 를 통해서 진행하였다.

10. Future Work

탈 중앙 검색엔진 서비스에 대한 홍보 및 이해를 증진하고 서비스 도입에 대한 고려를 돕는 것이 이 프로젝트의 목표이기 때문에, 기업 담당자와의 미팅을 통하여 게임 디자인의 완성도를 높이며 사용자 경험을 개선하는 작업을 진행할 것이다. 또한, 인트로 모듈, 역할 체험 모듈, 샌드박스 모듈, 크레딧 모듈 네 가지 기능을 agile 하게 개발할 것이다.

디자인 부분에서는 게임에 필요한 캐릭터, 버튼, 타이포그래피, 고도화된 레이아웃 등에 대하여 스케치 및 레퍼런스 스터디를 마무리하고, 기업 담당자와의 협의를 통하여 최종안을 확정하고자 한다. 이후 프론트에서는 JSX 문법으로 작성해 둔 각 컴포넌트들과 디자인 한 css 파일을 연결하면서 React Router 를 활용하여 각 view 를 게임 흐름에 맞춰 하나의 게임으로 구성해야한다. JavaScript 로 작성해둔 시뮬레이션 코드는 React 입장에서 외부 스크립트 파일이기 때문에 custom hook 및 useEffect()를 활용하여 이를 연결시켜야 한다. 이 때 샌드박스 페이지에서는 시뮬레이션이 한 번 돌아갈 때마다 페이지를 계속 렌더링해주어야하므로 React 컴포넌트의 수명주기를 적절히 관리해주어야 한다.

마지막 게임 테스트에 있어서는 여러 시나리오를 만들어 테스트할 예정이다. 특히 시뮬레이션에 있어서 비상식적인 값이 들어오거나 산출되는 경우가 있는지를 꼼꼼히 확인하여야 한다. 또한 서비스 검증에 있어, 앞서 말했듯이 이더리움 재단으로부터 실제 시뮬레이션 결과를 받아와야 한다. 이 부분을 기업 담당자님께 부탁드리고 그 결과값을 활용하여 우리가 구현한 시뮬레이션을 보완해 나가는 과정이 필요하다.

11. Division & Assignment of Work

항목	담당자
게임 설계	공은채, 문보설, 차우진, Charles Onyango
smart contract 스터디	Charles Onyango
시뮬레이션 코드 구현	문보설
UI 디자인 및 CSS	차우진
인트로 모듈 구현	공은채
크레딧 모듈 구현	공은채

역할체험 모듈 구현	공은채, 문보설, 차우진, Charles Onyango
샌드박스 모듈 구현	문보설
게임 테스팅	공은채, 차우진, 문보설, Charles Onyango
배포	Charles Onyango

12. Schedule

내용	9월			10월				11월				12월	
	2	3	4	1	2	3	4	1	2	3	4	1	2
벤치마크 분석	v	v	v										
smart contract 스터디	v	v	v	v	v								
오픈소스 코드 이해	v	v	v	v	v								
game flow 구체화	v	v	v	v	v								
wireframe 확정				v	v	v							
시뮬레이션 코드 구현					v	v	v						
인트로 화면 구현						v	v						
크레딧 화면 구현							v	v					
샌드박스 모드 구현							v	v					
기업 방문								v					
UI 디자인									v	v			
역할체험 모드 구현									v	v			
CSS 스타일링											v	v	
피드백 반영												v	v
앱 구동 테스트													v
최종 발표													v

◆ [Appendix] Detailed Implementation Spec

A. 인트로 모듈

a. Function nextPage()

버튼을 클릭하여 다음 화면으로 넘어가기 위해 사용되는 함수이다.

```
import { useHistory } from 'react-router-dom';

function nextPage() {
  let history = useHistory();

  const next = () => {
    history.push('/your-path')
  }

  return (
    <div>
      <button onClick={next}>Next</button>
    </div>
  )
}
```

B. 역할체험 모듈

a. Function nextPage()

버튼을 클릭하여 다음 화면으로 넘어가기 위해 사용되는 함수이다.

```
import { useHistory } from 'react-router-dom';

function nextPage() {
  let history = useHistory();

  const next = () => {
    history.push('/your-path')
  }

  return (
    <div>
      <button onClick={next}>Next</button>
```

```
    </div>
  )
}
```

b. Function userOrSuggester()

User인지 Suggester인지 사용자의 입력 값을 받아 적절한 화면을 렌더링하는 함수이다.

```
import React from "react";
import Select from "react-select";

function userOrSuggester() {
  const options = (
    () => [
      { value: 'user', label: 'User' },
                  { value: 'suggester', label: 'Suggester' },
    ],
    []
  );
  return (
    <div>
      <Select options={options} />
    </div>
  );
}

export default UserOrSuggester;
```

c. Function selectContent()

User에게 줄 contents를 여섯 개 중 하나로 결정한 사용자 값을 받는 함수이다.

```
import React from "react";
import Select from "react-select";

function selectContent() {
  const options = (
    () => [
      { value: 'content1', label: 'ContentBad' },
                  { value: 'content2', label: 'ContentTerrible' },
                  { value: 'content3', label: 'ContentNice' },
                        { value: 'content4', label: 'ContentFantastic' },
                        { value: 'content5', label: 'ContentGood' },
                        { value: 'content6', label: 'ContentVeryGood' },
```

```
    ],
    []
  );
  return (
    <div>
      <Select options={options} />
    </div>
  );
}

export default SelectContent;
```

d. Function selectSuggester()

User가 적절한 Suggester를 결정하는 부분으로 사용자가 결정한 입력 값을 받아 적절한 화면을 렌더링하는 함수이다.

```
Import React, {useState, useEffect} from “react”;

function checkSatisfaction() {
  const [count, setCheck] = useState(
    () => JSON.parse(window.localStorage.getItem("check")) || 0
  );

  useEffect(() => {
    window.localStorage.setItem("check", JSON.stringify(check));
  }, [check]);

  return (
                <button onClick={() => setCheck(check + 1)}>Satisfied</button>
                <button onClick={() => setCheck(check - 1)}>Dissatisfied</but-
ton>
        );
}

export default Check;
```

e. Function checkSatisfaction()

User가 만족/불만족을 결정한 사용자 입력 값을 받아오는 함수이다.

```
Import React, {useState, useEffect} from “react”;
```

```
function checkSatisfaction() {
  const [count, setCheck] = useState(
    () => JSON.parse(window.localStorage.getItem("check")) || 0
  );

  useEffect(() => {
    window.localStorage.setItem("check", JSON.stringify(check));
  }, [check]);

  return (
                <button onClick={() => setCheck(check + 1)}>Satisfied</button>
                <button onClick={() => setCheck(check - 1)}>Dissatisfied</but-
ton>
        );
}

export default Check;
```

C. 샌드박스 모듈 - 1. IO 모듈

a. Function nextPage()

버튼을 클릭하여 다음 화면으로 넘어가기 위해 사용되는 함수이다.

```
import { useHistory } from 'react-router-dom';

function nextPage() {
  let history = useHistory();

  const next = () => {
    history.push('/your-path')
  }

  return (
    <div>
      <button onClick={next}>Next</button>
    </div>
  )
}
```

b. Function ~update()

샌드박스 모드에서 시뮬레이션 코드가 돌아 간 뒤 시뮬레이션 코드의 실행 결과 변경된 UserList/SuggesterList/TotalAction/TotalReward/Deposit 객체를 input으로 받아 SimulationResult 객체에 반영하는 업데이트 함수이다.

```
Import React, {useState, useEffect} from “react”;

function suggestorListUpdate(SuggetorList sl, SimulationResult sr) {return;}
function userListUpdate(UserList ul, SimulationResult sr){return;}
function totalActionUpdate(TotalAction ta, SimulationResult sr) {return;}
function totalRewardUpdate(TotalReward tr, SimulationResult sr) {return;}
function depositUpdate(Deposit d, SimulationResult sr){return;}
```

c. Function showSimulation()

샌드박스 모드에서 업데이트된 시뮬레이션 결과를 내부적으로 반영한 뒤 변경된 simulationResult 값을 렌더링하는 함수이다.

```
Import React, {useState, useEffect} from “react”;
Import SuggesterList from "./SuggesterList";
Import UserList from "./UserList";
Import totalAction, totalReward, totalPunish from "./TotalFunctions";
Import deposit from "./Deposit";

function ShowSimulation(simulationResult sr) {

  useEffect(() => {
    window.localStorage.setItem("simulation", JSON.stringify(simulationRe-
sults));
  });

  return (
                <div>
                        <SuggesterList />
                        <UserList />
                        <totalAction />
                        <totalReward />
                        <totalPunish />
                        <deposit />
                </div>
        );
}
```

```
export default ShowSimulation;
```

d. Function importSimulation()

시뮬레이션 모듈이 있는 자바스크립트 파일을 가져와 실행할 수 있도록 하는 함수이다.

```
import { useEffect } from 'react';
const importSimulation = res=> {
  useEffect(() => {
    const script = document.createElement('script');
    script.src = res;
    script.async = true;
    document.body.appendChild(script);
return () => {
      document.body.removeChild(script);
    }
  }, [res]);
};
export default importSimulation;
```

D. 샌드박스 모듈 - 2. 시뮬레이션 모듈

a. Function SuggesterList(), Function UserList()

각각 SuggesterList객체, UserList 객체를 생성 및 관리하는 함수이다.

```
function SuggesterList(){}
function UserList(){}
```

b. Function Suggester()

Suggester객체를 생성 및 관리하는 함수이다.

c. Function getPunished()

Suggester가 처벌받을 때 deposit을 burn해야 하는 양만큼 감소시키고, 처벌 받은 횟수를 업데이트하는 역할을 한다.

d. Function updateService()

한 라운드가 끝나고 나서 Suggester의 처벌 받은 횟수에 따라 서비스 수준을 업데이트하는 역할을 한다.

```
function Suggester (rate, burn, service, deposit){
        this.id // 각 suggester 의 고유의 id 로, user 가 suggester 를 선택할 때 tie-breaking 용도로 사용한다.
        this.rate = rate
        this.burn = burn
        this.service = service
        this.deposit = deposit
        this.getPunished = function(){}
        this.updateService = function(){}
}
```

e. Function User()

User객체를 생성 및 관리하는 함수이다.

f. Function choice()

주어진 service가 tolerance보다 높은지를 기준으로 REWARD와 PUNISH를 결정한다. service가 user의 tolerance보다 높을 경우에는 REWARD 파라미터를 리턴하고, 아닌 경우 PUNISH 파라미터를 리턴한다.

g. Function getPunished()

처벌 상황에서 user의 deposit을 burn에 해당하는 만큼 감소시킨다.

```
function User (rate, burn, tolerance, deposit, payment){
        this.rate = rate
        this.burn = burn
        this.tolerance = tolerance
        this.deposit = deposit
        this.payment = payment
        this.choice = function(service){}
        this.getPunished = function(){}
}
```

h. Function getReward()

보상 상황에서 user 혹은 suggester의 deposit을 조정하는 역할을 한다. 해당 함수에 this로 지정되는 객체에 따라 deposit의 변화 방향이 달라진다. user 객체가 지정되는 경우, deposit은 user가 pay해야 하는 양 만큼 감소한다. 반면 suggester객체가 지정되는 경우 deposit은 user가 pay해야 하는 양 만큼 증가한다.

```
getReward = function(){}
```

i. Function isQuit()

한 거래가 발생한 후 각 agent가 거래를 끝낼 것인지를 확인하는 함수이다. 현재는 punish 혹은 reward 중 큰 값보다 deposit이 적어서 더 이상 거래를 지속할 수 없는 경우 중단 의사를 표현하도록 했다.

```
isQuit = function(){}
```

j. Function playOneGame()

user와 suggster간의 거래를 진행하는 함수이다. 해당 함수는 먼저 suggester의 service수준을 user.choice(suggester.service)로 user에게 전달한다. 그리고 이 함수의 리턴 값에 따라 적절한 코드를 수행한다. 리턴값이 PUNISH일 경우, user.getPunished()함수와 suggester.getPunished()함수를 호출한다. 이후 전체 처벌 횟수를 저장하는 totalPunishmen를 1 증가시킨다. 리턴값이 REWARD인 경우, getReward.call(user)와 getReward(suggester)를 호출한다. 그리고 전체 보상 횟수를 저장하는 totalReward를 1 증가시킨다. 리턴 값에 맞는 코드를 수행하고 나면, 거래 횟수를 저장하는 totalAction을 1 증가시킨다. 그리고 suggester.updateService()를 수행하여 suggester의 service 변수를 업데이트한다.

```
function playOneGame (user, suggester){}
```

k. Function playGames()

userList와 suggesterList를 가지고 전체 토너먼트를 수행하는 함수이다. 해당 함수는 먼저 userList의 각 user에 대해 suggesterList의 suggester들 중 거래하고자 하는 suggester를 선택하도록 한다. 거래할 suggester를 선택하는 기준은 두가지이다. 첫 번째로 rate가 user가 수용할 수 있

는 rate의 하한보다 높아야 한다. 다음으로 해당 suggester가 이미 다른 user에 의해 선택되지 않아야 한다(!suggester.opponent). 만약 이 조건을 만족시키는 suggester가 다수일 경우 rate가 높은 suggester로 tie-breaking한다. 그럼에도 다수의 suggester가 가능한 경우 그 중에서 id가 빠른 suggester를 선택한다. user가 suggester를 선택한 경우, 해당 suggester 객체에 opponent 변수를 추가하여 user를 저장한다. 마찬가지로 user의 opponent로 suggester를 저장한다. 다음으로 userList 중에서 opponent가 있는 user들에 대해 playOneGame(user, user.opponent)를 호출한다. 호출이 끝나면 isQuit.call(user)와 isQuit.call(suggester)를 수행한 후, 두 함수의 리턴 값이 모두 CONTINUE가 아닌 한 다시 playOneGame(user, user.opponent)을 호출한다. 함수가 종료되기 전에 ~update()함수들을 호출하여 결과값을 반영한다.

```
function playGames (userList, suggesterList){}
```

E. 크레딧 모듈

a. Function nextPage()

버튼을 클릭하여 다음 화면으로 넘어가기 위해 사용되는 함수이다.

```
import { useHistory } from 'react-router-dom';

function nextPage() {
  let history = useHistory();

  const next = () => {
    history.push('/your-path')
  }

  return (
    <div>
      <button onClick={next}>Next</button>
    </div>
  )
}
```

b. Function clickMoveUrl()

버튼을 클릭하여 url 링크가 있는 새 탭으로 페이지를 띄우는 목적으로 사용되는 함수이다.

```
import { react } from 'react';

function clickMoveUrl() {
  return (
    <div>
      <button onClick={() => window.open('[url link]', '_blank')}>
                                  [url link]
                        </button>
  )
}
```