프로젝트 최종 보고서

-소켓 통신을 사용한 1:1매칭 부루마블

팀원 명: 장은희, 유대균, 감달현

1:1 매칭 부루마블

1. 프로젝트 주제 및 계획

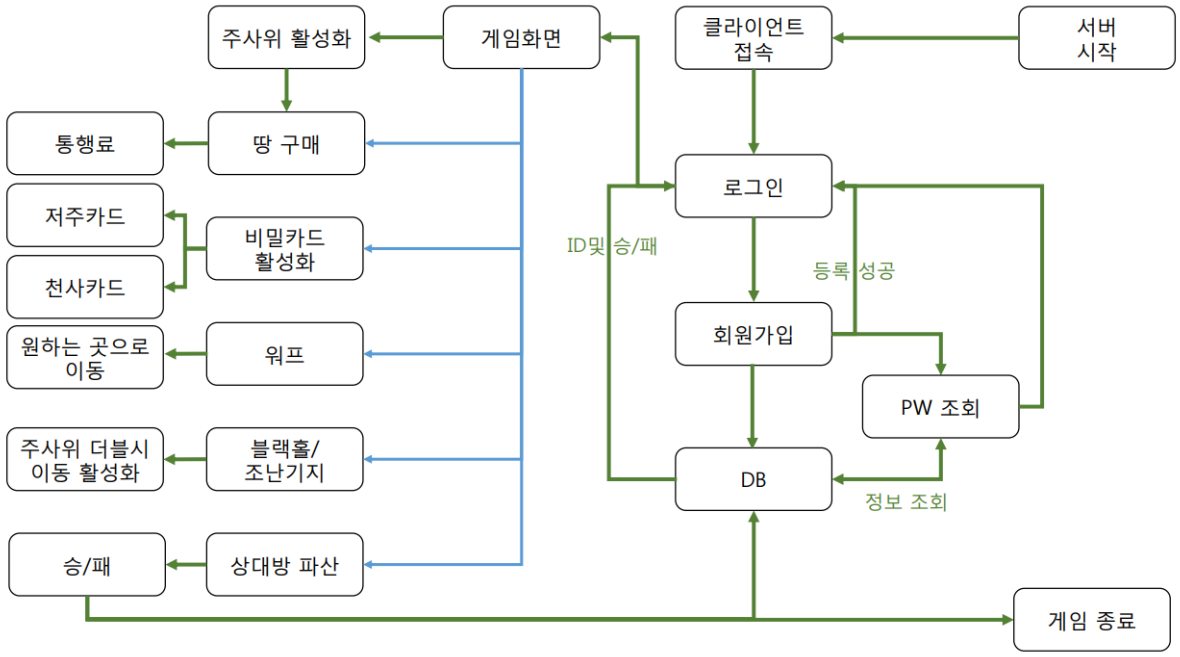
개발 기간	2021.03.19 ~ 2021.04.28 (총 40일)
개발 운영체제	Window
개발 도구	Eclipse
개발 언어	Java , Java FX, mySQL
개발 인원	3명
프로젝트 이름	大銀月(대은월)1:1매칭 부루마블 – 팀원의 이름을 한 글자씩 따서 지은 이름이다.
프로젝트 설명	전략 게임으로, 상대방을 먼저 파산시켜 승리하는 소켓 통신을 사용한 1:1매칭 부루마블 게임. 여러 명 접속이 가능하며, 접속을 한 클라이언트끼리 1:1매칭이 되어 게임을 시작한다. 서버와 클라이언트로 나뉘며, 서버에서는 클라이언트들의 정보를 수집하여, 데이터베이스에 저장한다. 클라이언트는 서버와 데이터를 송수신하여, 플레이 화면을 상대 플레이어와 함께 공유한다.
주제 선정 의도	즐겁게 프로젝트를 하며, 사용자에게도 즐거움을 줄 수 있는 주제 , 또한 mySQL을 이용하여 데이터를 저장하고, Java의 여러 클래스 및 메서드를 사용할 수 있는 주제로 게임이 적합하여, 게임을 주제로 선정함. 유저들에게 가장 접근하기 쉽고 많이 알려진 보드게임 부루마블을 토대로 제작하였다.
기능	1) 회원가입 2) 로그인 3) 랜덤 다이스 4) 땅 구매 5) 워프 6) 블랙홀 및 조난기지

	7) 말 이동 애니메이션 8) 랜덤 비밀카드 9) 승리 10) 항복 11) exe실행 파일

1.1 개발 시간표

	3.19	3.20	3.21	3.22	3.23	3.24	3.25	3.27	3.28	3.29	3.30	3.31	4.1	4.2	4.3
기획															
개발															
QA															
	4.4	4.5	4.6	4.7	4.8	4.9	4.10	4.11	4.12	4.13	4.14	4.15	4.16	4.17	4.18
기획															
개발															
QA															
	4.19	4.20	4.21	4.22	4.23	4.24	4.25	4.26	4.27	4.28					
기획															
개발															
QA															

1.2 플로우차트를 이용한 코드 설계



2 프로젝트 기획 및 기능

2.1 게임 규칙

2인용 전략 게임으로, 총 17개의 구매 가능한 땅, 7개의 비밀 카드 구역, 워프, 블랙홀, 조난기지, 그리고 시작지점인 지구가 존재한다. 선착순으로 플레이어의 순서를 정하며, 각 플레이어한테는 200만 원을 지급한다. 플레이어는 주사위를 돌려 눈금이 나온 수만큼 전진한다.

시작지점인 지구를 지나갈 때마다 수고비 20만 원을 지급받을 수 있다. 도착한 곳이 땅이면 플레이어는 구매를 할 수 있으며 구매 시 해당 땅의 가격만큼 돈이 차감된다. 비밀 카드는 총 30장의 다른 내용을 담고 있는 카드이며, 저주 카드와 축복 카드 두 부류로 나눌 수 있다. 저주 카드는 플레이어를 난감하게 만들고, 축복 카드는 플레이어를 도와주는 카드이다. 워프에 도착할 경우 플레이어는 가고 싶은 땅을 선택하여 갈 수 있다. 블랙홀과 조난기지에서는 플레이어의 주사위가 더블일 경우만 바로 빠져나올 수 있으며, 더블이 아닐 경우는 이동이 불가능해진다.

다른 플레이어가 구매한 땅에 도착할 경우 땅의 소유자한테 통행료를 지불해야 된다. 최종적으로 상대방을 먼저 파산시키는 플레이어가 승리한다.

2.2 기능

2.2.1 회원가입

신규 유저는 회원가입을 통해 로그인이 가능하다. 회원가입은 게임에서 사용한 이름, 로그인 시 필요한 아이디, 비밀번호가 필요하다. 위 세 개의 사항 중 하나라도 빈칸이 있으면"Please, fill in all the blanks" 문구와 함께 회원가입이 불가능해진다. 회원가입이 완료된 유저의 정보는 서버에 있는 데이터베이스에 저장된다.

2.2.2 로그인

회원가입을 통해 저장된 데이터베이스에서 아이디 및 비밀번호 정보를 불러오며, 정보가 일치하지 않을 경우 "Enter Correct ID/Password" 문구와 함께 로그인이 불가능해진다. 일치할 경우 "Login Successful…. Redirecting.." 문구가 뜨며, 게임 화면으로 진입한다.

2.2.3 랜덤 다이스

플레이어는 총 두 개의 주사위를 돌린다. Random 클래스를 활용하여 1부터 6까지 두 개의 랜덤숫자를 생성하여 더해준다. 더해준 수 만큼 플레이어의 말이 이동한다. 두 개의 주사위가 같은 숫자일 경우 플레이어에게는 한 번 더 주사위를 돌릴 기회가 주어진다.

2.2.4 땅 구매

ArrayList에는 땅의 이름, 소유자, 땅의 가격, 빌딩 건설 개수의 정보가 담겨있다. 플레이어는 도착한 곳의 땅을 구매할 수 있으며, 구매 시, ArrayList의 소유자 이름이 구매한 플레이어의 이름으로 바뀐다.

2.2.5 워프

워프에서는 자바FX ChoiceBox클래스를 통해 원하는 장소를 고를 수 있으며, 선택한 장소로 이동시켜준다.

2.2.6 블랙홀 및 조난기지

플레이어가 블랙홀 혹은 조난기지에 들어갈 경우, 블랙홀은 2턴 동안, 조난기지는 1턴 동안 이동이 불가능해진다. 블랙홀과 조난기지는 주사위가 더블이 나올 경우 탈출 할 수 있다. 또한 조난기지는 50만 원 돈을 차감한다.

2.2.7 말 이동 애니메이션

말 이동 애니메이션은 자바 FX의 Transition 클래스를 사용하였다. 주사위가 나온 숫자만큼 플레이어의 말은 해당 거리만큼 계산하여 움직인다.

2.2.8 랜덤 비밀 카드

총 30가지의 다른 기능을 가진 비밀 카드가 ArrayList 에 담겨있다. 카드는 Random 클래스를 이용하여 0부터 29까지 랜덤으로 숫자를 생성하여, 그 숫자에 해당하는 카드를 화면에 나타낸다.

1) 저주카드 시나리오

① 건물 철거

- 당신 땅의 모든 건물을 철거합니다. (해당 플레이어 땅의 모든 건물 철거)

② 은빛 저주

- Big Silver Moon의 저주를 풀기 위해 100만 원 헌금을 합니다. (100만 원 차감)

③ 달의 저주

- Big Silver Moon의 노여움을 받은 당신 조난 기지에서 시련을 받습니다. (해당 플레이어를 조난 기지로 이동시킴)

④ 大 저주

- 자만함으로 인해 Big Silver Moon의 노여움을 샀습니다. 이웃들에게 베푸는 모습으로 노여움을 푸십시오. (상대 플레이어에게 30만 원을 줌.)

⑤ 깜깜한 어둠

- 당신은 블랙홀에 갇히게 됩니다. (해당 플레이어 블랙 홀로 이동시킴.)

⑥ 외계인과의 전투

- 외계인과의 전투를 인해 50만 원의 손해를 봤습니다. (50만 원 차감.)

⑦ 모 아니면 도

- 당신의 땅과 상대편의 땅을 바꿉니다. (땅이 없을 시 무효.)

⑧ 고장난 우주선

- 우주선 수리 비용으로 50만 원을 지불합니다.

⑨ 사기꾼의 요구

- 우주여행 중 사기를 당한 당신, 가장 비싼 땅을 반액에 팔게 됩니다. (땅에 건물이 지어진 경우 건물도 반액에 처분.)

⑩ 우주의 저주

- 재수가 없는 당신, 우주의 신이 심심풀이로 당신에게 저주를 걸었습니다. 가지고 있는 모든 땅을 반납하세요. (해당 플레이어가 구매한 모든 땅 반납.)

⑪ 별과의 충돌

- 별과의 충돌로 인해 보유 금액의 50%를 손해 봅니다. (소유한 돈의 50% 차감.)

⑫ 우주주민의 박해

- 우주에 사는 주민이 이방인인 당신을 싫어합니다. 지구로 돌아가지만 수고비를 받지 못합니다. (해당 플레이어 지구도 이동, 수고비 20만 원 받지 못함.)

⑬ 쏟아지는 별똥별

- 별똥별로 인해 앞이 보이지 않습니다. 한 턴 쉽니다.

⑭ 외계인의 복수

- 과거 외계인을 약탈한 당신, 보유금액 모두 외계인에게 약탈당합니다.(해당 플레이어 모든 금액 차감.)

⑮ 축복 or 저주

- 10만 원을 내고 비밀 카드를 한 장 더 뽑습니다.

2) 축복카드 시나리오

① 앞으로 전진

- 5칸 더 앞으로 전진합니다.

② 달의 축복

- 당신은 Big Silver Moon에게 30만 원의 후원금을 받습니다. (해당 플레이어 30만 원 적립)

③ 은빛 축복

- Big Silver Moon이 당신의 조력자가 되어 상대방에게서 30만 원의 돈을 뺏어옵니다. (해당 플레이어 30만 원 적립, 상대 플레이어 30만 원 차감.)

④ 大 축복

- Big Silver Moon의 도움으로 타인의 땅을 뺏어옵니다.

⑤ 향수병

- 향수병으로 인해 지구로 돌아갑니다. (수고비를 받습니다. 20만 원 적립)

⑥ 원하는 땅을 찾아서

- 워프로 이동합니다. (원하는 땅 선택하여 이동.)

⑦ 은빛 축복

- 상대방은 당신에게 100만 원을 줘야 됩니다. (해당 플레이어 100만 원 적립, 상대 플레이어 100만 원 차감)

⑧ 달의 축복

- 우주 주민이 달의 축복을 받은 당신을 좋아합니다. 원하는 땅을 자신의 것으로 만듭니다. (상대방 땅 포함)

⑨ 大 축복

- 건물 두 개를 아무런 대가 없이 지을 수 있습니다. (해당 플레이어는 건물 두 개를 지을 자신의 땅을 선택함. 땅이 없을 시 카드 무효.)

⑩ 소소한 행복

- 우주 복권 4등 당첨! 10만 원을 받습니다. (해당 플레이어 10만 원 적립.)

⑪ 변수

- 상대방을 조난 기지에 보냅니다. (상대 플레이어 조난 기지로 이동.)

⑫ 무료 통행

- 통행료 없이 땅을 지나갑니다. (1회 사용할 수 있으며, 카드를 소지하고 있다가, 사용하고 싶을 때 사용 가능.)

⑬ 생일

- 상대방에게 20만 원씩 받습니다. (상대 플레이어 20만 원 차감, 해당 플레이어 20만 원 적립.)

⑭ 주사위 한번 더!

- 주사위를 한번 더 굴립니다.

⑮ 약탈

- 외계인 별에서 20만 원어치의 보물을 약탈합니다. (해당 플레이어 20만 원 적립.)

2.2.9 승리

1) 정상승리

상대 플레이어가 파산했을 경우 파산된 플레이어는 패+1, 승리한 플레이어는 승+1이 된다.

2) 비정상 승리

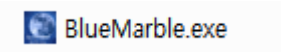
상대 플레이어가 비정상적인 종료를 수행할 경우, 사용자는 승리+1이 되며, 비정상 종료를 한 플레이어는 패+1이 된다.

2.2.10 항복

자신의 턴이 되면 항복 버튼이 활성화된다. 자신의 턴이 아닐 때는 항복이 불가능하다. 게임을 포기할 경우 플레이어는 패+1, 상대 플레이어는 승+1이 된다.

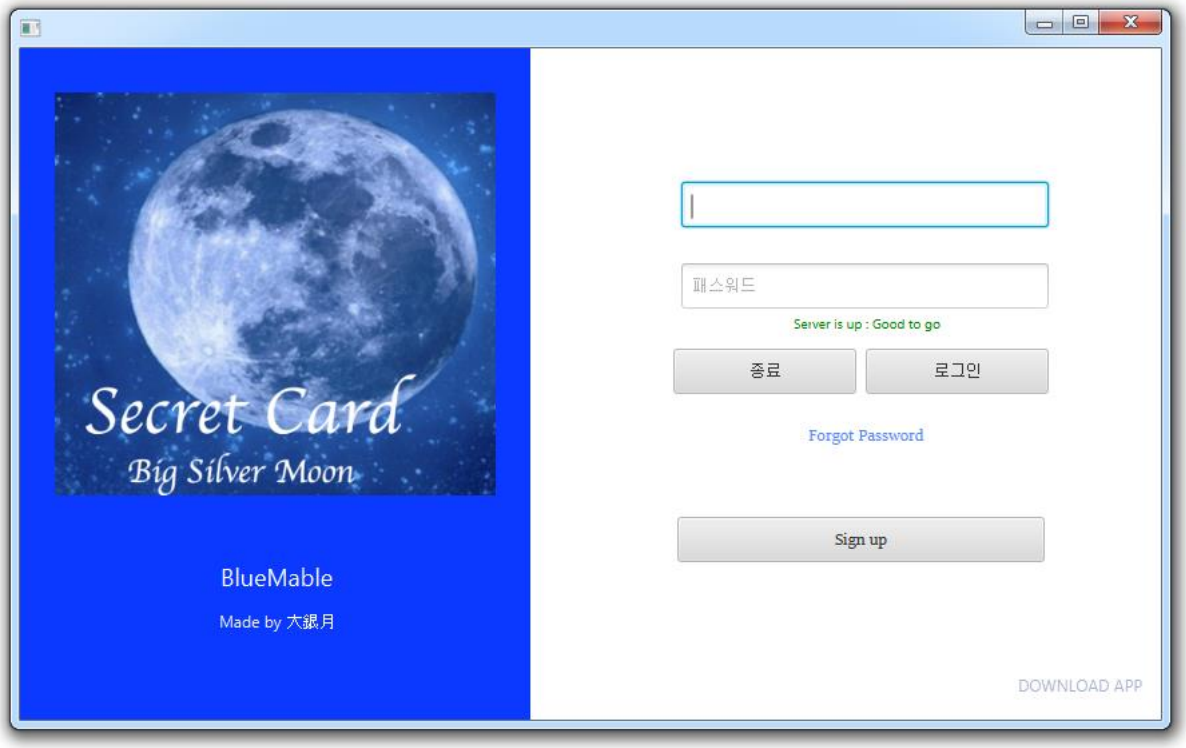
3. 사용자 인터페이스 및 매뉴얼

3.1 exe 실행파일



Exe 게임 실행파일, 사용자는 게임실행에 필요한 파일들이 담긴 압축파일 해제 후, BlueMarble.exe파일을 클릭하면 바로 게임을 실행할 수 있다. 클릭한 유저는 로그인 화면에 진입한다.

3.2 로그인 화면



사용자가 게임 접속을 시도할 시 나오는 로그인 화면. 사용자가 입력한 아이디와 비밀번호를 데이터베이스에서 찾고, 데이터베이스에 존재하지 않는 아이디 혹은 비밀번호를 입력 시, "Enter Correct ID/Password"가 나오며 로그인이 불가능 해진다.

비밀번호를 잊었을 시 "Forgot Password"를 통해 찾을 수 있다.

아이디와 비밀번호를 생성하지 않았을 경우 회원가입을 통해 아이디와 비밀번호를 생성해야 된다.

3.3 회원가입 화면


```
mysql> select * from member;
+----+------+------+------+------+------+
| no | ID   | pwd  | win  | lose | name |
+----+------+------+------+------+------+
|  1 | eun  | 1234 |    0 |    0 | eun  |
|  2 | hee  | 1234 |    0 |    0 | hee  |
+----+------+------+------+------+------+
```

회원가입 화면에서 사용자는 게임 내에서 사용할 이름과 로그인 시 필요한 아이디와 비밀번호를 생성할 수 있다. 생성한 정보는 데이터베이스에 저장된다. 또한 이미 존재하는 아이디일 경우 "ID already exits" 문구가 뜨며, 회원가입이 불가능해진다.

"Back" 버튼을 누를 경우 로그인 화면으로 돌아간다.

"Sign Up" 버튼을 누를 경우 빈칸이 있을 때는 "Please, fill in all the blanks" 문구가 뜨며 회원가입을 진행할 수 없으며, 모든 칸을 다 채웠을 경우 회원가입이 가능하다.

3.4 게임시작 전 게임 룰 설명화면

게임 시작 전, 모든 플레이어에게 게임 룰을 설명한다. 게임 룰은, 게임 화면 왼쪽 상단 "게임 설명서"를 클릭하면 계속 볼 수 있다.

3.5 게임시작 전 화면

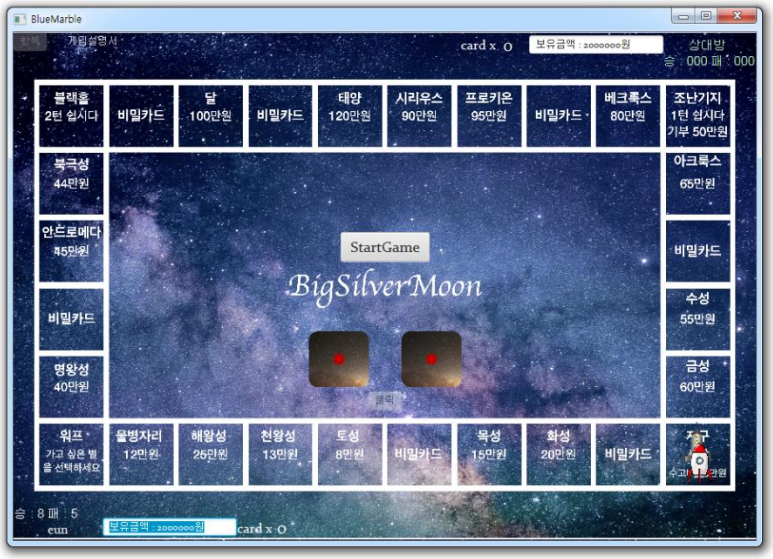


그림 1

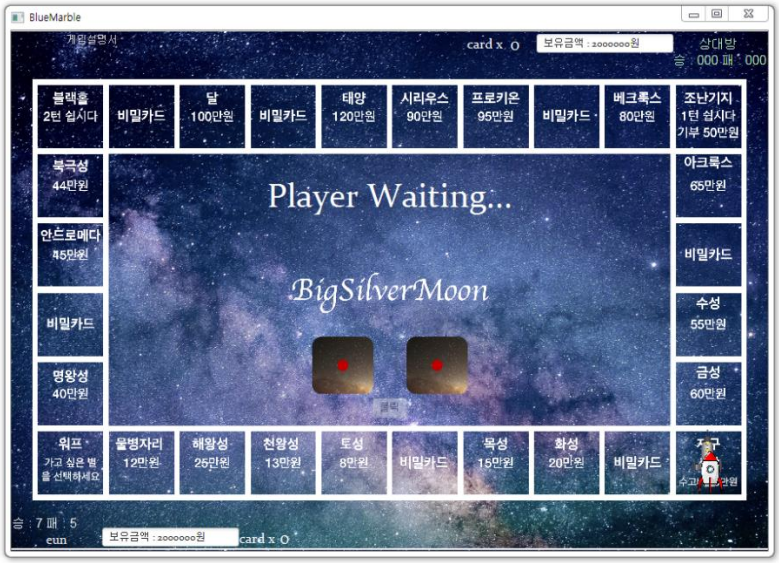


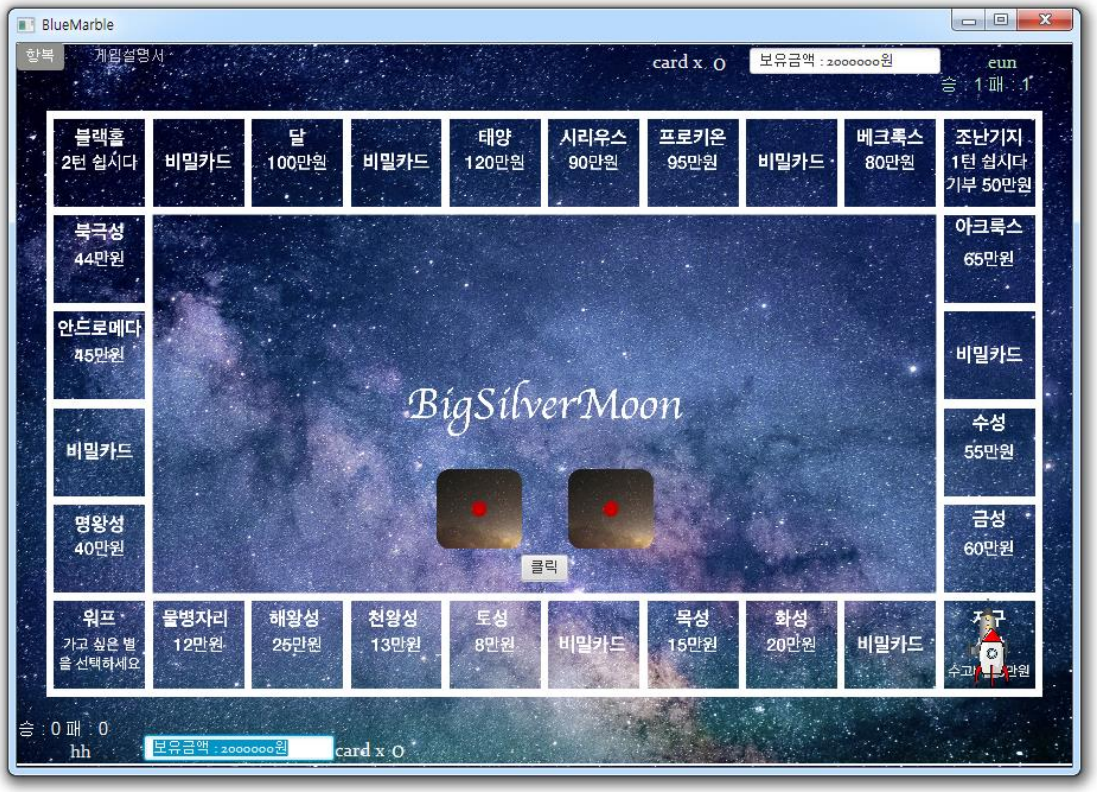
그림 2

사용자가 로그인 성공 시 볼 수 있는 화면이다.

"그림 1에서 "StartGame" 버튼을 누를 경우 사용자는 그림 2와 같이 플레이어 대기화면에 진입한다.

총 두 명의 사용자가 진행 할 수 있는 게임이므로, 상대 플레이어도 "StartGame"버튼을 눌러야지 본격적으로 게임을 시작할 수 있다.

3.6 게임 시작화면



총 두 명의 플레이어 모두 **"GameStart"**버튼을 누르면 주사위 클릭 버튼이 활성화된다. 게임 순서는 선착순이며, 먼저 들어온 사용자부터 주사위를 굴린다.

사용자 본인의 정보는 게임 화면 왼쪽 아래에 위치해 있으며, 상대 플레이어의 정보는 왼쪽 위 상단에 위치해 있다.

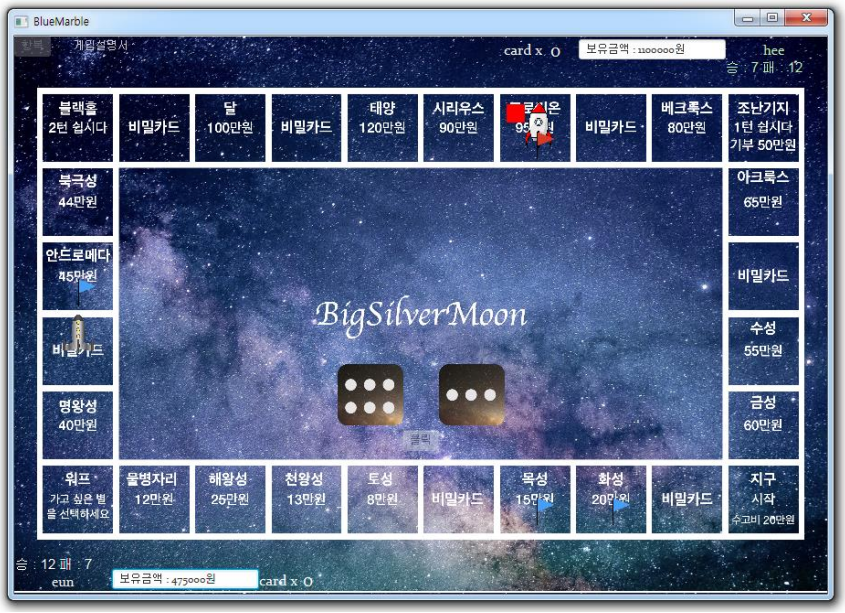
3.7 땅 도착 시 화면

사용자가 주사위를 굴리면, 사용자의 말은 주사위 눈금만큼 이동한다.

"구매" 버튼을 누를 경우 사용자는 구매 창 왼쪽 아래 **"구입 금액:"**에 쓰여있는 숫자만큼 보유금액을 차감한다. 구매완료 시, 구매 버튼이 사라진다.

사용자가 땅을 구매하고 싶지 않을 경우 **"확인"** 버튼을 눌러 구매 창을 패스한다.

3.8 땅 구매 완료 화면



사용자가 땅 구매 완료 시, 자신에게 해당하는 색깔의 깃발을 땅에 꽂아 표시한다.

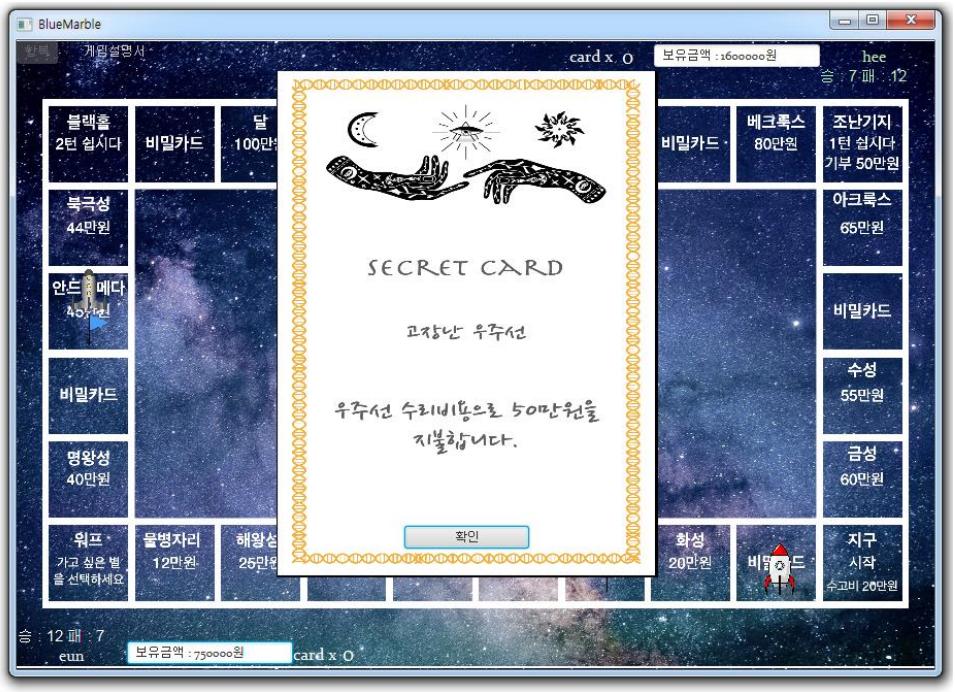
먼저 들어온 사용자는 파란 깃발을, 그 다음 들어온 사용자는 빨간 깃발을 사용한다.

사용자는 자신의 땅에 총 3개까지의 빌딩을 건설할 수 있다. 빌딩은 한 번에 하나만 지을 수 있으며, 깃발을 꽂고 난 후, 다음 턴에 자신의 땅에 도착했을 경우 건설이 가능하다.

3.9 통행료 지불 화면

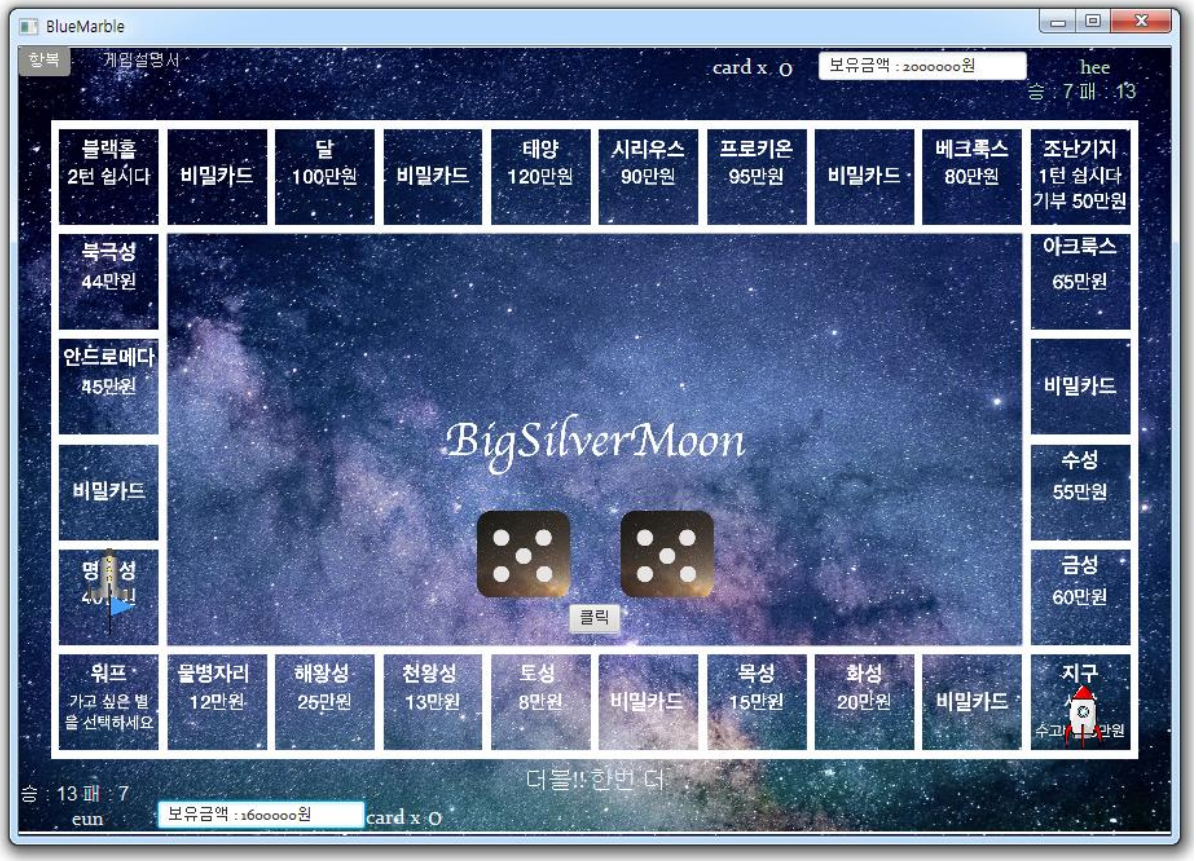
사용자가 상대 플레이어의 깃발이 꽂혀있는 땅에 도착했을 경우, 해당 땅값의 50% 통행료를 지불합니다. 만약 건물이 지어졌을 경우 건물 값 포함 50% 통행료를 지불하게 된다. 또한 보유금액이 충분할 경우, 통행료 지불 후, 상대 플레이어의 땅을 자신이 매수할 수 있다. 매수할 경우 깃발의 색깔은 자신의 깃발 색깔로 바뀐다.

3.10 비밀카드 발동 화면



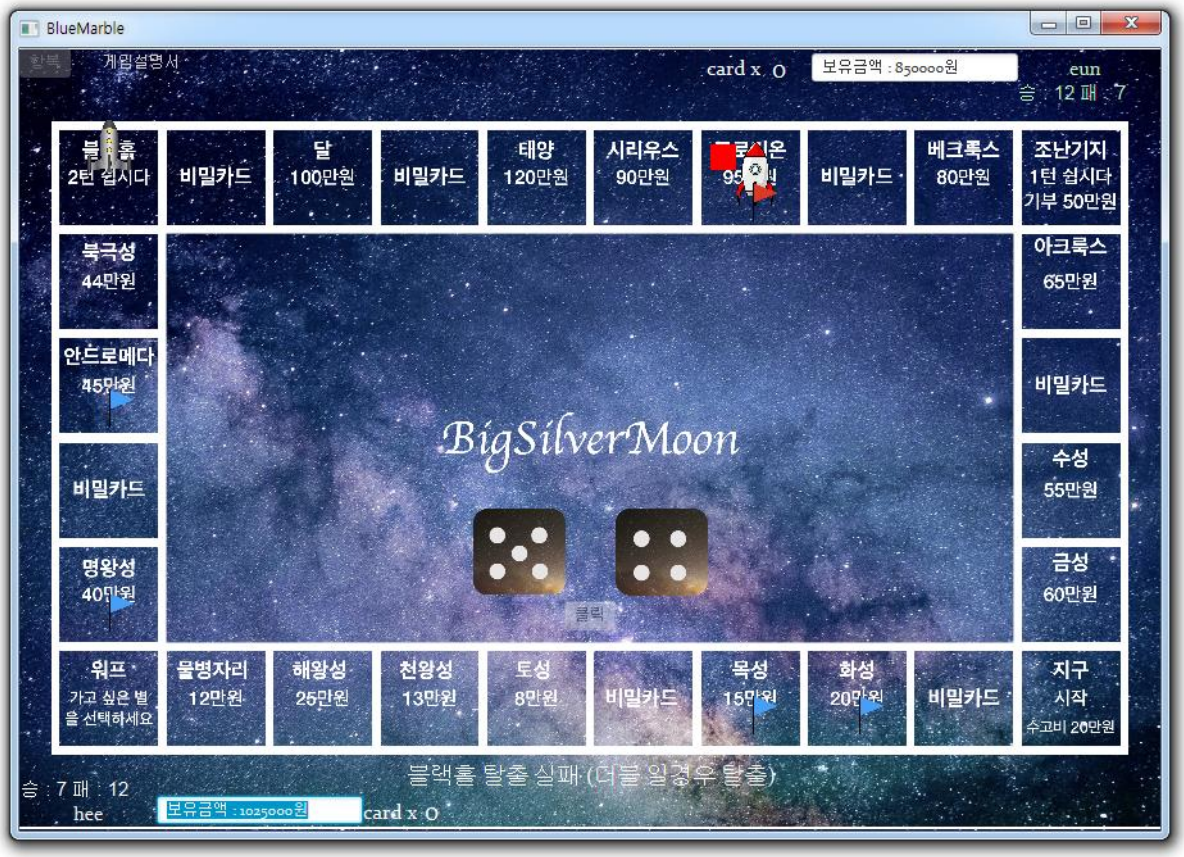
사용자가 "비밀 카드"에 도착했을 경우, 사용자와 상대플레이어 모두 비밀 카드의 내용을 볼 수 있다.

3.11 주사위 더블 화면



주사위가 똑같은 눈금이 나왔을 경우, 더블이라고 칭하며, 주사위를 한 번 더 굴릴 수 있는 기회를 준다. 더블이 나왔을 경우 사용자 화면 하단에는 "더블! 한번 더" 라는 문구가 뜬다.

3.12 블랙홀 탈출 실패 화면



사용자가 블랙홀에 도착했을 경우, 2턴을 쉬게 된다. 하지만 자신의 턴마다 주사위를 굴릴 수 있는데, 더블일 경우 바로 탈출이 가능하며, 더블이 아닐 경우 탈출을 실패한다. 조난 기지도 블랙홀과 같이 한 턴을 쉬고, 사용자 보유금액에서 50만 원을 차감한다.

3.13 워프

워프 지점에 도착하면, 사용자는 choice box에서 이동할 위치를 선택할 수 있다. 모든 곳이 선택 가능하며, 선택한 위치로 사용자는 바로 이동하게 된다. 아무것도 선택하지 않은 상태에서 선택을 누르면 창이 닫히지 않는다.

3.14 승리화면


```
mysql> select * from member;
+----+------+------+------+------+------+
| no | ID   | pwd  | win  | lose | name |
+----+------+------+------+------+------+
|  1 | eun  | 1234 |    1 |    0 | eun  |
|  2 | hee  | 1234 |    0 |    1 | hee  |
+----+------+------+------+------+------+
```

1) 정상 승리: 보유금액이 마이너스가 됐을 경우, 게임은 종료되며, 마이너스가 아닌 사용자가 승리하게 된다.

2) 항복: 항복 버튼을 누른 플레이어는 패배를 하게 된다.

3) 비정상 종료: 비정상적인 종료를 실행할 경우 패배를 하게 된다.

승리를 하면 "승:1 패:0"으로 승리 횟수가 올라가게 된다. 이 정보는 데이터베이스에 기록되며 로그인 시 기록은 계속 남아있다.

4. 프로그램 구성 및 코드 설명

4.1 프로그램 구성

4.1.1 BlueMarbleServer

DBConn 클래스	**[MySQL접속 클래스]** -connectDB() 메소드 : DB접속. -disconnectDB() 메소드 : DB접속 종료
Server 클래스	**[소켓 서버 클래스]** -Start() 메소드 : 서버 시작. -msgSendAll(String msg) : 접속중인 모든 Client에 메시지를 보내기 위한 함수. -msgSend(String msg, BlueMarbleThread thread) : 특정 Client에 메시지를 보내기 위한 함수. -machtingPlayer() : Client에서 Start버튼 클릭 시 매칭 시켜주는 함수.(queue 이용). -BlueMarbleThread 클래스 : 클라이언트 접속 시 실행되는 클래스, Thread를 상속받아 실행가능. -run() - while문을 통해 Client로부터 받은 요청을 조건에 맞게 체크하고 실행한다. -dbWork(String menu, String sql, String tmsg) – client로부터 받은 DB요청으로 상황에 맞게 처리한다.

4.1.2 BlueMarble

Main 클래스	**[Application초기화 및 실행]** -init() 메소드 : Application초기화 -start() 메소드 : Application 시작 -stop() 메소드 : 매칭이 되었는지 서버에 확인 요청 하고 Application 종료.
PlayerClient 클래스	**[서버로 소켓 접속을 위한 클래스]** -connectServer() 메소드 : 서버 ip 와 8888포트로 접속.
LoginController 클래스	**[로그인 관련 기능을 컨트롤하는 클래스]** -handleButtonAction(MouseEvent event) 메소드 : 버튼이벤트 처리 함수. -handleLabelAction(MouseEvent event) 메소드 : 라벨이벤트 처리함수. -forgotPW() 메소드 : 비밀번호 찾기 메소드.

	-signup() 메소드 : 회원가입 메소드. -getMsg() 메소드 : 서버에서 메시지를 받는 함수. 메시지 별 실행. -gameRule() 메소드 : 게임 룰을 표시하기 위한 함수. -logIn() 메소드 : 로그인 메소드
AppController 클래스	**[게임 화면 관련 기능을 컨트롤하는 클래스]** -handleLabelAction(MouseEvent event) 메소드 : 라벨이벤트 메소드. -gameRule() 메소드 : 라벨이벤트 동작시 실행 하는 메소드. 게임 룰 화면을 표시. -initialize(URL location, ResourceBundle resources) 메소드 : AppContorller 시작시 초기화. -setLogin(String login) 메소드 : 로그인 설정 함수. -StartGame() 메소드 : StartGame 버튼 클릭 시 동작하는 함수. -gameFinish(String winner) 메소드 : 게임 종료 시 동작하는 함수. -getMsg() 메소드 : 소켓 메세지를 받는 메소드, 메시지에 따른 동작 실행. -setMoney(int my, int your) 메소드 : 상황별 Money변화를 셋팅하는 메소드 -rollTheDice() 메소드 : 주사위 굴리기 버튼 클릭시 시작하는 함수. -movePiece(int num, PlayerData player) 메소드 : 말을 움직이는 메소드. -setDice() 메소드 : 주사위 랜덤 셋팅. -setDiceImage() 메소드 : 주사위 값에 맞는 이미지로 설정 -showDialog() 메소드 : 이동이 끝난 위치의 정보를 표시. -setSecretCard() 메소드 : 비밀 카드 초기 설정, 비밀 카드를 다 뽑으면 다시 실행. -setSecretCardNum() 메소드 : 비밀 카드 랜덤으로 뽑는 메소드. -showSecretCard(int num, int actionTurn) 메소드 : 뽑힌 비밀 번호 카드 보여주는 메소드 -actionSecretCard(int num) 메소드 : 뽑힌 비밀 카드의 번호에 따른 동작 실행. pass() 메소드 : 비밀 카드 통행료 없이 지나가기가 걸렸을 때 동작 하는 메소드. -oneMore() 메소드 : 비밀 카드 한번 더 가 나왔을 때 동작하

	는 메소드. -takeLand(int index) 메소드 : 비밀 카드로 땅 관련 동작이 나왔을 때 실행되는 메소드. -changeLand(ChoiceBox<String> choiceYourBox, ChoiceBox<String> choiceMyBox) 메소드 : 비밀카드로 땅 교환이 나왔을 때 동작하는 메소드. -getChoice(ChoiceBox<String> choiceBox, int index) 메소드 : 비밀카드로 땅 선택시 동작하는 메소드. -SetFlag(int landPosition, PlayerData player) 메소드 : Flag 설치 시 동작 하는 메소드 -SetBuilding(int landPosition, int located, PlayerData player) 메소드 : Building 설치 시 동작 하는 메소드. -ChangeLandFlag(int landPosition, PlayerData player1, int changePosition, PlayerData player2) 메소드 : 상대방의 땅과 나의 땅을 변경할 때 사용되어지는 메소드. -BuyLand(String name, String owner, int price) 메소드 : 땅 구매시 사용되는 메소드. -setPosition(PlayerData player) 메소드 : 말의 이동 애니메이션을 위해 말의 이동 경로를 설정하는 메소드. -setPosition2(int move, PlayerData player) : 비밀 카드 상황에 따른 말의 이동 경로를 설정하는 메소드 -setWarp(AnchorPane anchorPane, Stage dialog) 메소드 : 워프 설정 메소드. -getChoiceWarp(ChoiceBox<String> choiceBox, Stage dialog) 메소드 : 워프 이동 메소드

4.2 Thread 사용 설명서

4.2.1 JavaFX Application Thread

Application 의 launch() 메서드를 호출하면서 생성된 JavaFX Application Thread는

1. start() 메서드를 실행시키면서 모든 UI를 생성한다.

2. 컨트롤에서 이벤트가 발생할 경우 컨트롤러의 이벤트 처리 메서드를 실행한다.

4.2.2 JavaFX Application Thread 사용시 주의사항

JavaFX Application Thread가 시간을 필요로 하는 작업은 되도록 하지 않아야 된다..

시간을 필요로 하는 작업을 하게 되면 그 시간 동안 UI는 반응하지 않고 멈춰있는 상태가 되므로 다른 작업 스레드를 생성해서 처리해야 한다.

예를 들어 말의 움직임을 애니메이션으로 이동 시키는 작업은 시간이 필요한 작업이기 때문에 스레드 없이 동작 시, 정상적인 동작을 하지 않게 된다. 따라서 작업 스레드에서 처리하도록 해야 하지만, 작업 스레드가 직접 UI를 변경 할 수 없다.

그러므로 Platform.runLater()를 사용해 UI작업을 처리 한다.

4.2.3 Platform.runLater()

작업 스레드가 직접 UI를 변경하는 것이 불가능하므로, UI 변경이 필요하면 작업 스레드는 UI 변경 코드를 Runnable로 생성하고 이 것을 매개 값으로 Platform의 runLater() 정적 메서드를 호출한다.

runLater()는 이벤트 큐에 Runnable을 저장하고 바로 리턴 한다. 이벤트 큐에 저장된 Runnable들은 저장된 순서에 따라 JavaFX Application Thread에 의해 하나씩 처리되며, UI 변경 작업을 수행한다.

runLater()는 지연 처리 라는 뜻으로, 주어진 매개 값을 Runnable이 바로 처리하는 것이 아니라 이벤트 큐에 먼저 저장된 Runnable을 처리한 후 처리하는 방식이다.

4.3 코드 설명

code	해설

[Server]
`public class Server {`
`  private ServerSocket ss = null;`
`  private Socket s = null;`
 	
`  String id,pw,name;`
`  int cnt = 0;`
`  Queue<BlueMarbleThread> player = new LinkedList<>();`
`  ArrayList<BlueMarbleThread> playerThreads = new ArrayList<BlueMarbleThread>();`
 	
`  public void start() {`
`  try {`
 	
`  ss = new ServerSocket(8888); // 소켓 활성화`
`  System.out.println("server start");`
`  machtingPlayer();// 매칭 Thread 실행`
`  while (true) {`
`  s = ss.accept(); //클라이언트 접속`
 	
`  BlueMarbleThread p = new BlueMarbleThread(); // 접속된 클라이언트 서버 처리 스레드 생성`
 	
`  playerThreads.add(p); //접속 리스트`
 	
`  p.start(); //클라이언트 서버 처리 스레드 시작`
`  }`
`  } catch (Exception e) {`
`  System.out.println("[Multi Server]start() Exception 발생!!");`
`  }`
`  }`
 	
`  public static void main(String[] args) {`
`  Server server = new Server();`
`  server.start(); //서버 시작`
`  }`
	서버 구성 및 시작
	[서버 시작]
	1. 서버를 실행하면 포트 8888을 통해 Client에서 Sever로 접속을 확인.
	2. 서버에 접속한 클라이언트는 독립된 처리 Thread에서 작동.
	3. 접속된 Client를 ArrayList로 저장 관리

[LoginController]

```
  } else if (event.getSource() == btnSignup) { //회원가입
      signup();
  }
```

```
TextField name_signup = (TextField) anchorPane.lookup("#name_signup"); // 이름 입력 버튼
TextField id_signup = (TextField) anchorPane.lookup("#id_signup"); //아이디 입력 버튼
TextField pw_signup = (TextField) anchorPane.lookup("#pw_signup"); //패스워드 입력 버튼
Button signup_btn = (Button) anchorPane.lookup("#signup_btn"); //회원가입 완료 버튼
Button back_btn = (Button) anchorPane.lookup("#back_btn"); //뒤로가기 버튼
Label lblErrors = (Label) anchorPane.lookup("#lblErrors"); //에러 표시 라벨

//회원가입 완료 버튼 클릭 이벤트
signup_btn.setOnAction((EventHandler<ActionEvent>) new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
        Platform.runLater(() -> {

            String name = name_signup.getText(); //입력된 이름을 가져옴
            String id = id_signup.getText(); //입력된 아이디를 가져옴
            String pw = pw_signup.getText(); // 입력된 비밀번호를 가져옴

            String sql="select count(id) as cnt from member where id ='"+id+"'";
            if (name.isEmpty() || id.isEmpty() || pw.isEmpty()) { //빈칸이 있는지 체크
                lblErrors.setTextFill(Color.RED);
                lblErrors.setText("Please, fill in all the blanks");
            } else { //빈칸 없이 입력이 되었을뜨 동작
                //쿼리를 서버에 보내 DB에 요청
                SocketConnect.getOutMsg().println("Signup/" + sql+"/"+id+"/"+pw+"/"+name);
                signup.close();
            }

        });
    }
});
```

[Server]

```
case "Signup": // 회원 가입
    int cnt = 0; //중복 확인 변수
    dbConn.rs = dbConn.state.executeQuery(sql);
    if (dbConn.rs.next()) {
        cnt = dbConn.rs.getInt("cnt"); //DB조회
    }
    if (cnt == 0) { //아이디 중복 X
        sql = "insert into member values(null, '" + id + "','" + pw + "'," + 0 + "," + 0 + ",'" + name
                + "')";
        dbConn.state.executeUpdate(sql);

    } else {//아이디 중복시
        tmsg = "Signup/" + id;
        msgSend(tmsg, BlueMarbleThread.this);
    }

    break;
```

[LoginController]

```
}else if(id.equals("Signup")) { //회원가입 중복시 동작
    Platform.runLater(()->{
        setLblError(Color.TOMATO, "ID already exists.");
    });
```

회원가입

[동작]

1. LoginController 에서 ButtonEvent 처리를 통해 회원가입 기능 시작한다.
2. 이름, 아이디, 비밀번호를 다 입력 했는지 확인후 서버에 요청한다.
3. 중복 확인후 중복이 아닐 경우 회원가입 완료. 중복일 경우 요청한 Client 에게 메시지 전송.
4. Client는 중복 메시지를 받아 ID already exists를 화면에 표시한다.

[설명]

회원가입은 게임에서 사용할 이름, 로그인 시 필요한 아이디, 비밀번호가 필요하다. 위 세 개의 사항 중 하나라도 빈칸이 있으면"Please, fill in all the blanks" 문구와 함께 회원가입이 불가능해진다. 중복된 아이디일 경우 ID already exists. 문구와 함께 회원가입이 불가능해진다. 회원가입이 완료된 유저의 정보는 서버에 있는 데이터베이스에 저장된다.

[LoginController]

```
if (event.getSource() == btnSignin) { //로그인
    logIn();
    // login here
```

```
// 로그인 체크
private void logIn() {
    String id = txtUsername.getText(); //입력된 아이디 가져옴
    String pwd = txtPassword.getText(); // 입력된 패스워드 가져옴
    System.out.println(id + "," + pwd);
    if (id.isEmpty() || pwd.isEmpty()) { //아이디와 패스워드 입력이 다 되어있는지 체크
        setLblError(Color.TOMATO, "Empty credentials"); //빈칸이 있을때
        id = "Error";
    } else {//빈칸이 없을때
        // query
        String sql = "SELECT id,WIN,LOSE FROM MEMBER Where id = '" + id + "' and pwd = '" + pwd + "'";
        SocketConnect.getOutMsg().println("Login/" + sql); //쿼리를 서버에 보내 DB에 요청

    }

}
```

[Server]

```
case "Login":
    dbConn.rs = dbConn.state.executeQuery(sql); //Client로 부터 받아온 쿼리를 db에 요청
    if (!dbConn.rs.next()) { //쿼리 결과가 없을 경우
        tmsg = "Error"; //에러메시지를 클라이언트로 보냄
    } else { //쿼리결과가 있을경우 id,win,lose 정보를 클라이언트로 보냄
        tmsg = dbConn.rs.getString(1) + "/" + dbConn.rs.getInt(2) + "/" + dbConn.rs.getInt(3);
        myPlayer = new Player(dbConn.rs.getString(1), dbConn.rs.getInt(2), dbConn.rs.getInt(3));
    }
    msgSend(tmsg, BlueMarbleThread.this);
    break;
```

[AppController]

```
// setLogin -> 로그인시 동작 함수
public void setLogin(String login) {
   String[] rmsg = null;
   String msg = login;
   rmsg = msg.split("/");
   myPlayer.setId(rmsg[0]);
   myPlayer.setWin(Integer.parseInt(rmsg[1]));
   myPlayer.setLose(Integer.parseInt(rmsg[2]));
   Platform.runLater(() -> {
      myName.setText(myPlayer.getId());
      myWinLose.setText("승 : " + myPlayer.getWin() + " 패 : " + myPlayer.getLose());
   });

}
```

로그인

[동작]

1. LoginController 에서 ButtonEvent 처리를 통해 로그인 기능을 시작한다.
2. 이름, 아이디 입력 했는지 확인 후 Server 에 요청한다.
3. 서버는 받은 요청을 DB 로 보낸다.
4. 로그인 실패 시 error 메시지를 보내고, 성공 시 id, win ,lose 정보를 Client 로 보낸다.
5. LoginController 에서 AppController 의 setLogin 메소드를 실행.

[설명]

회원가입을 통해 저장된 데이터베이스에서 아이디 및 비밀번호 정보를 불러오며, 정보가 일치하지 않을 경우 "Enter Correct ID/Password" 문구와 함께 로그인이 불가능해진다. 일치할 경우 "Login Successful.... Redirecting.." 문구가 뜨며, 게임 화면으로 진입한다.

[LoginController]

```
@FXML
public void handleLabelAction(MouseEvent event) throws SQLException { //라벨 이벤트
    if (event.getSource() == btnForgot) {
        forgotPW();
    }
}
public void forgotPW() { //비밀번호 찾기
    AnchorPane anchorPane = null;
    Stage signup = new Stage(StageStyle.UTILITY);
    signup.initModality(Modality.WINDOW_MODAL);
    signup.initOwner(primaryStage);
    signup.setTitle("Find Password");

    try {
        anchorPane = (AnchorPane) FXMLLoader.load(getClass().getResource("/fxml/signup.fxml"));

    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    TextField name_signup = (TextField) anchorPane.lookup("#name_signup");// 이름 입력 필드
    TextField id_signup = (TextField) anchorPane.lookup("#id_signup");// 아이디 입력 필드
    TextField pw_signup = (TextField) anchorPane.lookup("#pw_signup");// 패스워드 입력필드
    Button signup_btn = (Button) anchorPane.lookup("#signup_btn"); //찾기 버튼
    signup_btn.setText("Find");//찾기버튼 텍스트 설정
    Button back_btn = (Button) anchorPane.lookup("#back_btn");//뒤로가기 버튼
    Label lblErrors = (Label) anchorPane.lookup("#lblErrors");// 에러표시 라벨
    Label title = (Label) anchorPane.lookup("#title"); // 타이틀
    Label pw = (Label) anchorPane.lookup("#pw"); //비밀번호
    title.setText("Find PW"); //타이틀 텍스트 설정
    pw.setVisible(false);
    pw_signup.setVisible(false);

    signup_btn.setOnAction((EventHandler<ActionEvent>) new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent e) {
            Platform.runLater(() -> {

                String name = name_signup.getText(); //입력된 이름 가져옴
                String id = id_signup.getText(); // 입력된 아이디 가져옴
                String sql = "SELECT pwd FROM MEMBER Where id = '" + id + "' and name = '" + name + "'";
                if (name.isEmpty() || id.isEmpty()) { //빈칸이 있을때
                    lblErrors.setTextFill(Color.RED);
                    lblErrors.setText("Please, fill in all the blanks");//에러표시
                } else {//빈칸 없을때

                    SocketConnect.getOutMsg().println("FindPW/" + sql); //서버에 쿼리 요청
                    signup.close();
                }

            });
        }
    });

    back_btn.setOnAction(event -> signup.close());
    Scene scene = new Scene(anchorPane);
    signup.setScene(scene);
    signup.show();
}
```

[server]

```
case "FindPW"://비밀번호 찾기
    dbConn.rs = dbConn.state.executeQuery(sql);
    tmsg = "";
    System.out.println(dbConn.rs);
    if (!dbConn.rs.next()) {
        tmsg = "FindPW/NotFound"; //비밀번호 찾기 실패시
    } else {
        tmsg = "FindPW/" + dbConn.rs.getString(1);//비밀번호 찾기 성공시
        System.out.println("pw>>>>>" + tmsg);
    }
    msgSend(tmsg, BlueMarbleThread.this);
    break;
```

```
// 패스워드 찾기
else if(id.equals("FindPW")) {
    System.out.println("pw>>>>>"+rmsg[1]);
    findPw = rmsg[1];
    Platform.runLater(() ->{
        Stage dialog = new Stage(StageStyle.UTILITY);
        dialog.initModality(Modality.WINDOW_MODAL);
        dialog.initOwner(primaryStage);
        dialog.setTitle("패스워드 확인");
        AnchorPane anchorPane = null;

        try {
            anchorPane = (AnchorPane) FXMLLoader.load(getClass().getResource("/fxml/Message.fxml"));
        } catch (IOException e2) {
            e2.printStackTrace();
        }

        Label Message = (Label) anchorPane.lookup("#Message");
        if(findPw.equals("NotFound")) {
            Message.setText("존재하지 않는 사용자 입니다.");
        }else {
            String pw = "찾으시는 패스워드는 '"+ findPw +"' 입니다.";
            Message.setText(pw);
        }

        Button Complete = (Button) anchorPane.lookup("#Complete");
        Complete.setOnAction(event -> dialog.close());

        Scene scene = new Scene(anchorPane);
        dialog.setScene(scene);
        dialog.show();
    });
```

비밀번호 찾기

[동작]

1. 로그인화면에서 비밀번호 찾기 클릭 시 forgotPW() 메소드 실행.
2. 서버를 통해 DB에 요청. 아이디와 이름을 보낸다.
3. 서버에서 DB로 쿼리 요청 후 일치하는 값이 없으면 Client에 에러를 전송.
4. 서버에서 DB로 쿼리 요청 후 일치하는 값이 있으면 Client에 비밀번호를 전송.
5. LoginController에서 메시지를 받아 화면에 표시.

[AppController]

```
StartGame.setOnAction((event) -> StartGame());
```

```
// StartGame ->버튼 눌렀을때 동작함수
public void StartGame() {
    Thread thread = new Thread() {
        @Override
        public void run() {
            int cnt = 0;
            String text[] = { "Player Waiting", "Player Waiting.", "Player Waiting..", "Player Waiting...",
                    "Player Waiting....", "Player Waiting...." };
            //서버에 게임준비 완료 메시지를 보냄
            SocketConnect.getOutMsg()
                    .println("Ready/" + myPlayer.getId() + "/" + myPlayer.getWin() + "/" + myPlayer.getLose());
            Platform.runLater(() -> {
                StartGame.setVisible(false);
                btnGiveup.setVisible(false);
                Waiting.setVisible(true);
            });

            while (true) { //서버가 매칭을 잡아 줄때 까지 동작
                if (ready == 1) { //서버에서 ready 값을 받아와 1 이되면 종료
                    Platform.runLater(() -> {
                        Waiting.setText("Game Start!"); // 게임 시작 메시지 표시
                        btnGiveup.setVisible(true);
                    });
                    try {
                        Thread.sleep(1000); //1초동안 보여줌
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    Platform.runLater(() -> {
                        Waiting.setVisible(false);
                    });

                    break;
                }
                int t = cnt++;
                Platform.runLater(() -> {
                    Waiting.setText(text[t]); //매칭대기 메시지 표시
                });

                if (cnt == 6) {
                    cnt = 0;
                }
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    };
    thread.setDaemon(true);
    thread.start();

}
```

[server]

```
} else if (rmsg[0].equals("Ready")) {
    ready = rmsg[1];
    player.offer(BlueMarbleThread.this);
```

```
void machtingPlayer() {
    Thread machting = new Thread() {
        @Override
        public void run() {
            while (true) {
                if (player.size() == 2) {
                    BlueMarbleThread player1 = player.poll(); //첫번째 플레이어 데이터
                    BlueMarbleThread player2 = player.poll(); //두번째 플레이어 데이터
                    player1.yourPlayer = player2;
                    player2.yourPlayer = player1;
                    //상대방정보와 턴값 설정
                    String p1msg = "Start/" + player2.myPlayer.username + "/" + player2.myPlayer.win + "/"
                            + player2.myPlayer.lose + "/" + 0;
                    String p2msg = "Start/" + player1.myPlayer.username + "/" + player1.myPlayer.win + "/"
                            + player1.myPlayer.lose + "/" + 1;
                    msgSend(p1msg, player1); //상대방 정보와 턴값을 플레이어 1번에게 보냄
                    msgSend(p2msg, player2); //상대방 정보와 턴값을 플레이어 2번에게 보냄
                }
                try {
                    Thread.sleep(1);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    };
    machting.start();
}
if (rmsg[0].equals("Start")) { //메세지 태그가 Start 일때 동작
    yourPlayer.setId(rmsg[1]); // 상대방 아이디 저장
    yourPlayer.setWin(Integer.parseInt(rmsg[2])); //상대방 승리 저장
    yourPlayer.setLose(Integer.parseInt(rmsg[3]));// 상대방 패배 저장
    myPlayer.setTurn(Integer.parseInt(rmsg[4])); // 나의 턴 저장
    Platform.runLater(() -> {
        yourName.setText(yourPlayer.getId());
        yourWinLose.setText("승 : " + yourPlayer.getWin() + " 패 : " + yourPlayer.getLose());

    });

    yourPlayer.setTurn((myPlayer.getTurn() + 1) % 2);
    // user[myPlayer.getTurn()] = id;
    // user[yourPlayer.getTurn()] = tname;
    // System.out.println("user[0] : " + user[0] + " user[1] : " + user[1]);
    if (myPlayer.getTurn() == 0) {
        btn1.setDisable(false);
        btnGiveup.setDisable(false);
    }
    ready = 1;//매칭 ready값 변경
}
```

게임 스타트 및 매칭

[동작]

1. 게임화면에서 GameStart 버튼 클릭 시 매칭 대기 상태로 넘어가고 화면에 표시
2. 서버 player(Queue)에 대기열 추가.
3. machtingPlayer Thread에서 매칭 인원 체크 2명일 때 매칭 성공. 매칭 성공 메시지 보냄.
4. AppController에서 대기 상태를 멈추고 상대방 정보를 설정, 게임 시작

[설명]

플레이어가 게임 시작 버튼을 누르면 매칭 대기 상태가 되고 매칭에 성공하면 상대방 정보를 받아와 화면에 나타내고 게임이 시작된다. 먼저 게임 버튼을 누른 사람이 첫 번째 턴이 된다.

[rollTheDice]

```
public void rollTheDice() {
   Double.setVisible(false);
   Thread thread = new Thread() {
      @Override
      public void run() {
         Platform.runLater(() -> {
            btn1.setDisable(true);//주사위 버튼 비활성화
            btnGiveup.setDisable(true); //포기버튼 비활성화
         });
         Platform.runLater(() -> {
            setDice();   //주사위 랜덤 설정
         });
         Platform.runLater(() -> {
            setDiceImage(); //변경된 주사위값으로 이미지 설정
         });
         try {
            Thread.sleep(50);
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
         if (myPlayer.getBre() != 0) { //실값이 있을때
            if (d1 != d2) {  //더블이 아닐때
               Platform.runLater(() -> {
                  Double.setText(planetData.get(myPlayer.getPosition()).getName() + " 탈출 실패 (더블 일경우 탈출)");
                  Double.setVisible(true);
               });
               myPlayer.bre--; //실카운트 하나 감소
               turn = yourPlayer.getTurn(); //상대턴으로 변경
               SocketConnect.getOutMsg().println("ChangeTurn/" + turn); //변경된 턴값 서버를 통해 상대에게 보냄
               return;
            } else {//더블일때
               myPlayer.bre = 0; //실값 초기화
               Platform.runLater(() -> {
                  Double.setText("탈출 성공 ! 더블!! 한번 더");
                  Double.setVisible(true);
               });
            }
         } else {//실값이 0일때
            if (d1 != d2) { //더블이 아닐때
               turn = yourPlayer.getTurn(); //턴값 변경
            } else { //더블일 경우
               Platform.runLater(() -> {
                  Double.setText("더블!! 한번 더");
                  Double.setVisible(true);
               });
            }
         }
         SocketConnect.getOutMsg().println("Dice/" + d1 + "/" + d2); //주사위 클릭값을 서버를통해 상대에게 보냄
           Platform.runLater(() -> {
               System.out.println("d1 : " + d1 + " d2 : " + d2);
               movePiece(d1 + d2, myPlayer); //말을 움직임
            });
            try {
               Thread.sleep(150 + 50 * (d1 + d2) + 100);
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
            Platform.runLater(() -> {
               if (myPlayer.getCard() > 0) { //패스 카드가 있으면
                   //나의땅이 아닐때
                  if (planetData.get(myPlayer.getPosition()).getOwner().equals(yourPlayer.getId())) {
                     pass();
                  } else { //나의땅일때
                     showDialog();
                  }
               } else {//패스 카드 없을때
                  showDialog();
               }
            });
         }

      };
      thread.setDaemon(true);
      thread.start();
   }
```

랜덤 다이스

[동작]

1. 주사위 버튼 클릭 시 rollTheDice 메소드가 실행.
2. 클릭 버튼과 항복 버튼을 비활성화 하고 setDice 메소드를 통해 주사위 2 개의 값을 랜덤으로 설정.
3. 주사위 값에 맞춰 setDiceImage 메소드를 통해 주사위 이미지 설정.
4. 주사위 던져야 되는 횟수, 더블 등의 값들을 체크하고 서버를 통해 설정된 주사위 값을 상대에게 전송.
5. 설정된 주사위 값만큼 말을 이동.
6. 말 이동이 끝나면 현재 위치 정보를 showDialog 메소드를 통해 표시.
7. 상대는 getMsg 메소드에서 메시지를 받아 말을 움직임. 다이얼로그 X.

[설명]

플레이어는 총 두 개의 주사위를 돌린다. Random 클래스를 활용하여 1부터 6까지 랜덤숫자를 생성하여 주사위 값을 정해 합한 수 만큼 플레이어의 말이 이동한다. 주사위 수가 같을 경우 플레이어에게는 한 번 더 주사위를 돌릴 기회가 주어진다.

```
            Platform.runLater(() -> {
               System.out.println("d1 : " + d1 + " d2 : " + d2);
               movePiece(d1 + d2, myPlayer); //말을 움직임
            });
            try {
               Thread.sleep(150 + 50 * (d1 + d2) + 100);
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
            Platform.runLater(() -> {
               if (myPlayer.getCard() > 0) { //패스 카드가 있으면
                  //나의땅이 아닐때
                  if (planetData.get(myPlayer.getPosition()).getOwner().equals(yourPlayer.getId())) {
                     pass();
                  } else { //나의땅일때
                     showDialog();
                  }
               } else {//패스 카드 없을때
                  showDialog();
               }
            });
         }

      };
      thread.setDaemon(true);
      thread.start();
   }
// SocketConnect.getOutMsg().println("Dice/" "/" + d1 + "/" + d2);
else if (rmsg[0].equals("Dice")) { //메세지 태그가 Dice일때 동작
   d1 = Integer.parseInt(rmsg[1]); //주사위 1번값
   d2 = Integer.parseInt(rmsg[2]); //주사위 2번값
   System.out.println("내턴 아님...");
   Platform.runLater(() -> {
      setDiceImage(); //받아온 주사위값으로 이미지 셋팅
      movePiece(d1 + d2, yourPlayer); //받아온 주사위1,2 값 만큼 말 이동
   });

   System.out.println("현재턴:" + turn);

}
```

[BuyLand(String name, String owner, int price)]

```
// 땅구매시 사용되는 함수
public void BuyLand(String name, String owner, int price) {

    Stage dialog = new Stage(StageStyle.UTILITY);
    dialog.initModality(Modality.WINDOW_MODAL);
    dialog.initOwner(primaryStage);
    dialog.setTitle("구매 확인");
    int cost = 0;
    //코스트 설정
    if (planetData.get(myPlayer.getPosition()).getCount() == 0) {
       cost = price;
    } else {
       cost = price + (price / 2) * (planetData.get(myPlayer.getPosition()).getCount() - 1);
    }

    AnchorPane anchorPane = null;

    try {
       anchorPane = (AnchorPane) FXMLLoader.load(getClass().getResource("/fxml/Message.fxml"));
    } catch (IOException e2) {
       e2.printStackTrace();
    }

    Label Message = (Label) anchorPane.lookup("#Message");
    curMoney = (Label) anchorPane.lookup("#curMoney");

    Button Complete = (Button) anchorPane.lookup("#Complete");

    Complete.setOnAction(event -> dialog.close());
//        Rectangle rec[] = new Rectangle[2];

//        ImageView flag = new ImageView();
    System.out.println("cost : " + cost);
    System.out.println("price : " + price);
    if (myPlayer.getMoney() < cost) { //보유금액과 코스트 비교, 돈이 부족할때
       Message.setText("보유금액이 부족합니다.");
    } else {//돈이 충분할때
       if (planetData.get(myPlayer.getPosition()).getOwner().equals("X")) {
          planetData.get(myPlayer.getPosition()).setOwner(myPlayer.getId());
          PlanetOwner.setText(myPlayer.getId());
       }
       if (!planetData.get(myPlayer.getPosition()).getOwner().equals(myPlayer.getId())) { // 내땅이 아닐경우
          ChangeLandFlag(myPlayer.getPosition(), myPlayer, -1, null); // 땅 가져올때 플래그를 변경 함수
          setMoney(-cost, cost);
          PlanetOwner.setText(myPlayer.getId());
          Message.setText("구매 완료하였습니다.");
       } else { // 빈땅 이거나가 내땅인경우
          if (planetData.get(myPlayer.getPosition()).getCount() == 0) { // 빈땅일때
             setFlag(myPlayer.getPosition(), myPlayer);
             Message.setText("구매 완료하였습니다.");
             if (owner.equals("X")) {
                setMoney(-price, 0);
             } else {
                setMoney(-cost, 0);
             }

          } else if (planetData.get(myPlayer.getPosition()).getCount() > 3) {
             Alert alert = new Alert(AlertType.WARNING);
             alert.setTitle("알림");
             alert.setHeaderText("구매 불가능");
             alert.setContentText("이미 지을 수 있는 빌딩 개수의최대입니다.");
             alert.showAndWait();
             cnt1 = 0;
             cnt2 = 0;
             return;
          } else {
             int locate[] = { -33, -10, 13 };
             setBuilding(myPlayer.getPosition(), locate[planetData.get(myPlayer.getPosition()).getCount() - 1],
                   myPlayer); // 건물 설치
             // 함수 호출
             Message.setText("구매 완료하였습니다.");
             setMoney(-(price / 2), 0);
          }
       }

    }
```

[AppController()의 getMsg()]

```
// 소켓
// SocketConnect.getOutMsg().println("SetFlag/"+position[turn]+"/"+turn);
else if (rmsg[0].equals("SetFlag")) {//메세지 태그가 SetFlage 일때 동작
    int tp = Integer.parseInt(rmsg[1]); //Flag를 설치할 위치 설정
    int tt = Integer.parseInt(rmsg[2]); //Flag의 색깔 설정
    PlayerData player;
    if (tt == myPlayer.getTurn()) {
       player = myPlayer;
    } else {
       player = yourPlayer;
    }
    Platform.runLater(() -> {
       setFlag(tp, player); //Flag 설치
    });

}
// 소켓
// SocketConnect.getOutMsg()
// .println("SetBuilding/" + position[turn] +usernum+"/"+
// locate);
else if (rmsg[0].equals("SetBuilding")) {//메세지 태그가 SetBuilding 일때 동작
    int position = Integer.parseInt(rmsg[1]); //설치할 위치 설정
    int curturn = Integer.parseInt(rmsg[2]); //텐값에 따른 색깔 설정
    int locate = Integer.parseInt(rmsg[3]); // 1,2,3 번째에 맞는 위치 설정
    PlayerData player;
    if (curturn == myPlayer.getTurn()) {
       player = myPlayer;
    } else {
       player = yourPlayer;
    }
    Platform.runLater(() -> {
       setBuilding(position, locate, player);//빌딩 설치

    });

}
```

땅 구매

[동작]

1. 땅 구매 버튼 클릭 시 BuyLand() 메소드 실행.
2. 구매 가격과 현재 보유 금액을 체크하여 구매 성공 여부 결정.
3. 주인이 없는 경우 setFlag() 메소드를 통해 flag 를 설치한다.
4. 내 땅인 경우 setBuilding() 메소드를 통해 building 을 설치한다.
5. 내 땅이 아닌경우 ChangeLandFlag() 메소드를 통해 주인을 바꾸고 건물의 색상을 변경한다.
6. 변경된 정보는 서버를 통해 상대에게 전달. getMsg()메소드에서 상대도 같은 변경을 하게 한다.

[설명]

ArrayList에는 땅의 이름, 소유자, 땅의 가격, 빌딩 건설 개수의 정보가 담겨있다. 플레이어는 도착한 곳의 땅을 구매할 수 있으며, 구매 시, ArrayList의 소유자 이름이 구매한 플레이어의 이름으로 바뀐다.

[setWarp(AnchorPane anchorPane, Stage dialog)]

```
// 워프 다이얼로그 함수
public void setWarp(AnchorPane anchorPane, Stage dialog) {
    // btn1.setDisable(true);
    Button button = new Button();
    Label label = new Label();
    ChoiceBox<String> choiceBox = new ChoiceBox<String>();
    label.setText("이동할 위치 선택 : ");
    label.setLayoutX(170);
    label.setLayoutY(200);

    choiceBox.setLayoutX(280);
    choiceBox.setLayoutY(195);
    choiceBox.setPrefWidth(100);

    int cnt = 1;
    for (int i = 0; i < planetData.size(); i++) {
        if (planetData.get(i).getName().equals("비밀카드")) {
            choiceBox.getItems().add(planetData.get(i).getName() + cnt++);
        } else if (planetData.get(i).getName().equals("워프")) {
            choiceBox.getItems().add(planetData.get(i).getName() + " 이동X");
        } else {
            choiceBox.getItems().add(planetData.get(i).getName());
        }

    }
    choiceBox.setValue("워프 이동X");
    button.setLayoutX(160);
    button.setLayoutY(250);
    button.setPrefWidth(100);
    button.setText("선택");
    anchorPane.getChildren().add(button);
    anchorPane.getChildren().add(choiceBox);
    anchorPane.getChildren().add(label);
    button.setOnAction(event -> getChoiceWarp(choiceBox, dialog));

}
```

```
// 워프 동작 함수
private void getChoiceWarp(ChoiceBox<String> choiceBox, Stage dialog) {
    int index = choiceBox.getItems().indexOf(choiceBox.getValue());
    int move = 0;
    //워프로 선택한 위치까지의 거리 계산
    if (index >= myPlayer.getPosition()) {
        move = index - myPlayer.getPosition();
    } else {
        move = 19 + index;
    }
    int movecount = move; //거리 셋팅
    System.out.println(move);
    dialog.close();
    System.out.println(myPlayer.getPosition());
    Thread thread = new Thread() {
        @Override
        public void run() {
            path = new Path();
            Platform.runLater(() -> {
                //상대방에게 말의 움직임을 서버를 통해 알려줌
                SocketConnect.getOutMsg().println("MovePiece/" + myPlayer.getTurn() + "/" + movecount);
                for (int i = 0; i < movecount; i++) {
                    setPosition(myPlayer);//움직임
                }
                pathTransition.setDuration(Duration.millis(150 + 50 * (movecount)));
                pathTransition.setPath(path);
                if (myPlayer.getTurn() == 0) {
                    pathTransition.setNode(C1);
                } else {
                    pathTransition.setNode(C2);
                }
                pathTransition.play();

            });
            try {
                Thread.sleep(150 + 50 * (movecount) + 300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            //워프 이동시 다이얼로그 표시
            Platform.runLater(() -> {

                showDialog();
            });

        }

    };
    thread.setDaemon(true);
    thread.start();

}
```

[상대 AppController getMsg()]

```
// SocketConnect.getOutMsg().println("MovePiece/"+ movedice + "/" +
// movecount);
else if (rmsg[0].equals("MovePiece")) { //메세지 태그가 MovePiece 일때 동작
    int movedice = Integer.parseInt(rmsg[1]); //움직일 말 설정
    int movecount = Integer.parseInt(rmsg[2]); // 움직이는 카운트 설정
    PlayerData movePlayer;
    if (movedice == myPlayer.getTurn()) {
        movePlayer = myPlayer;
    } else {
        movePlayer = yourPlayer;
    }
    Platform.runLater(() -> {
        movePiece(movecount, movePlayer); //말 움직임
    });

}
```

워프

[동작]

1. 이동 위치가 워프일 때 setWarp 메소드를 통해 다이얼로그를 띄운다.
2. 이동할 위치를 선택하고 나면, getChoiceWarp 메소드를 통해 말이 이동.
3. 이동한 데이터를 서버를 통해 상대에게 전달.
4. 상대는 getMsg 메소드를 통해 말의 이동을 실행.

[설명]

워프에서는 자바 FX ChoiceBox 클래스를 통해 원하는 장소를 고를 수 있으며, 선택한 장소로 이동시켜준다.

[AppController()의 showDialog()]

```
//특정 위치 정보 셋팅
if (name.equals("워프")) {
    setWarp(anchorPane, dialog2);//워프함수 동작
    Complete.setVisible(false);
} else if (name.equals("조난기지")) {
    setMoney(-500000, 0);
    myPlayer.bre = 1; //쉼값
    turn = yourPlayer.getTurn();
} else if (name.equals("블랙홀")) {
    myPlayer.bre = 2; //쉼값
    turn = yourPlayer.getTurn();
} else {
    Complete.setVisible(true);
}
```

블랙홀 및 조난기지

[동작]

1. 조난 기지나 블랙홀에 들어갈 경우 쉼 값을 설정한다. 주사위 쉬는 값(쉼값/bre)은 턴 변경 시 이용 되는 변수

[설명]

플레이어가 블랙홀 혹은 조난기지에 들어갈 경우, 블랙홀은 2턴 동안, 조난기지는 1턴 동안 이동이 불가능해진다. 블랙홀과 조난기지는 주사위가 더블이 나올 경우 탈출 할 수 있다. 또한 조난기지는 50만 원 돈을 차감한다.

[movePiece(int num, PlayerData player)]

```
// movePiece 말의 움직임이 있을 때 동작
public void movePiece(int num, PlayerData player) {
    Thread thread = new Thread() {
        @Override
        public void run() {
            path = new Path(); //새로운 패스 설정
            Platform.runLater(() -> {
                for (int i = 0; i < num; i++) {
                    setPosition(player); //패스 경로 설정
                }
                pathTransition.setDuration(Duration.millis(150 + 50 * num));
                pathTransition.setPath(path);
                if (player.getTurn() == 0) { //움직일 말 선택
                    pathTransition.setNode(C1);
                } else {
                    pathTransition.setNode(C2);
                }
                pathTransition.play(); //말 움직임

            });
            try {
                Thread.sleep(150 + 50 * num + 300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    };
    thread.setDaemon(true);
    thread.start();
}
```

말 이동 애니메이션

[동작]

1. 말의 이동 횟수 만큼 setPosition 메소드를 통해 경로를 저장하여 말을 움직인다.
2. 움직일 말을 턴 값에 따라 결정한다.

[설명]

말 이동 애니메이션은 자바 FX의 Transition 클래스를 사용하였다. 주사위가 나온 숫자만큼 플레이어의 말은 해당 거리만큼 계산하여 움직인다.

[showSecretCard(int num, int actionTurn)]

```
public void showSecretCard(int num, int actionTurn) {
    System.out.println("num : " + num);
    int cardNum;
    if (secretCardList.size() == 0) { // 모든 비밀카드가 오픈되고 나면 비밀카드 다시 채워줌
        setSecretCard();
        cardNum = secretCardList.get(num);
        secretCardList.remove(num);
    } else {
        cardNum = secretCardList.get(num);
        secretCardList.remove(num);
    }
    // 테스트용
    // cardNum = 23;
    System.out.println("cardNum : " + cardNum);
    System.out.println("secretCardList size : " + secretCardList.size());
    AnchorPane anchorPane = null;
    Stage dialog = new Stage(StageStyle.UTILITY);
    try {
        anchorPane = (AnchorPane) FXMLLoader.load(getClass().getResource("/fxml/secretCard.fxml"));
    } catch (IOException e) {
        e.printStackTrace();
    }
    ImageView secretCard = (ImageView) anchorPane.lookup("#secretCard");
    String imgName = cardNum + ".png";
    Image img = new Image("/images/secretCard/" + imgName);
    secretCard.setImage(img);
    Button Complete = (Button) anchorPane.lookup("#Complete");
    // int tmp = cardNum;
    Complete.setOnAction(event -> {
        dialog.close();
        // 5 7 14 16 17 24 29
        if (myPlayer.getTurn() == actionTurn && cardNum != 17 && cardNum != 29 && cardNum != 3 && cardNum != 9
                && cardNum != 19 && cardNum != 20) {
            SocketConnect.getOutMsg().println("ChangeTurn/" + turn);
        }
        checkdialog.remove(dialog);

    });
    dialog.setAlwaysOnTop(true);
    if (myPlayer.getTurn() == actionTurn) {
        actionSecretCard(cardNum);
    }
    Scene scene = new Scene(anchorPane);
    dialog.setScene(scene);
    dialog.setX(primaryStage.getX() + 260);
    dialog.setY(primaryStage.getY() + 59.5);
    dialog.show();
    checkdialog.add(dialog);
}
```

[상대 AppController의 getMsg()]

```
// 비밀카드 보여주기
else if (rmsg[0].equals("ShowSecretCard")) {
    int tturn = Integer.parseInt(rmsg[1]);
    int num = Integer.parseInt(rmsg[2]);
    Platform.runLater(() -> {
        showSecretCard(num, tturn);
    });

}
```

랜덤 비밀 카드

[동작]

1. 비밀 카드에 들어갈 경우 showSecretCard 메소드가 실행된다.
2. setSecretCard 메소드를 통해 비밀 카드를 설정하고 랜덤으로 숫자를 받아온다.
3. 받아온 숫자에 맞는 비밀 카드를 보여주고 자기턴일때는 비밀 카드가 동작 하도록 한다.
4. 상대방에게 서버를 통해 현재 뽑힌 비밀 카드 정보를 보낸다.
5. 상대방은 getMsg 메소드에서 showSecretCard 를 호출한다. 자기 턴이 아니기 때문에 동작하지 않음.

[설명]

총 30가지의 다른 기능을 가진 비밀 카드가 ArrayList 에 담겨있다. 카드는 Random 클래스를 이용하여 0부터 29까지 랜덤으로 숫자를 생성하여, 그 숫자에 해당하는 카드를 화면에 나타낸다.

[AppController()의 actionSecretCard()]

```
case 0: // 소유한 본인땅의 모든 건물 삭제

    for (int i = 0; i < planetData.size(); i++) {
        if (myPlayer.getId().equals(planetData.get(i).getOwner())) {
            for (int j = 1; j < planetData.get(i).getBuilding().size(); j++) {
                SocketConnect.getOutMsg().println("removeFlag/" + i + "/" + 1);
            }
        }
    }

    break;
```

저주카드 시나리오 - 건물철거

[동작]

1. planetData에 저장 되어 있는 각 땅의 소유주 이름과 자신의 아이디를 eqauls문으로 비교하며, 같으면 전부 remove로 빌딩을 지운다.

[AppController()의 actionSecretCard()]

```
case 4:// money[myPlayer.getTurn()] -= 1000000;
    setMoney(-1000000, 0);
    break;
```

[상대방 AppController 의 getMsg()]

```
else if (rmsg[0].equals("Money")) {// 메세지 태그가 Money 일때 동작
    yourPlayer.setMoney(Integer.parseInt(rmsg[1])); // 상대 머니 설정
    myPlayer.setMoney(Integer.parseInt(rmsg[2])); // 나의 머니 설정
    Platform.runLater(() -> {
        MyMoney.setText("보유금액 : " + myPlayer.getMoney() + "원"); // 나의 머니 표시
        YourMoney.setText("보유금액 : " + yourPlayer.getMoney() + "원");// 상대의 머니 표시
    });
}
```

저주카드 시나리오 – 은빛 저주

[동작]

1. setMoney() 메소드를 통해 100만원을 자신의 소유한 돈에서 차감한다. 상대방 화면에서도, 돈 100만원이 차감되게 보인다.

[AppController()의 actionSecretCard()]

```
case 5://조난기지로 이동
    myPlayer.setBre(1);
    Cardposi = 1;
    int posi2 = myPlayer.getPosition();
    turn = yourPlayer.getTurn();
    setPosition2(posi2, myPlayer);
    break;
```

저주카드 시나리오 – 달의 저주

[동작]

1. setPosition()메소드에서 자신이 있는 자리부터 조난기지 까지의 거리를 계산하여, 자신의 말을 조난기지에 보낸다.

[AppController()의 actionSecretCard()]

```
case 6:
    setMoney(-300000, 300000);
    // 상대편 금액 +30만원 로직 추가해야함. socket으로 +30만원 전송
    break;
```

저주카드 시나리오 - 大 저주

[동작]

1. setMoney메소드로 자신은 30만원 차감, 상대방은 +30만원 차감을 한다.

[AppController()의 actionSecretCard()]

```
case 11:// 가장 비싼 땅을 반액에 팔음. 건들이 지어진 경우 반액에 처분.
    int max = 0;
    int tmp = 0;
    for (int i = 0; i < planetData.size(); i++) {
        if (myPlayer.getId().equals(planetData.get(i).getOwner())) {
            max = (max > planetData.get(i).getPrice()) ? max : planetData.get(i).getPrice();
        }

    }
    for (int i = 0; i < planetData.size(); i++) {
        if (myPlayer.getId().equals(planetData.get(i).getOwner())) {
            for (int j = 0; j < planetData.get(i).getBuilding().size(); j++) {
                if (max == planetData.get(i).getPrice()) {
                    SocketConnect.getOutMsg().println("removeFlag/" + i + "/" + 0);
                    if (tmp == 0) {
                        int tmoney = ((max + ((max / 2) * (planetData.get(i).getBuilding().size() - 1))) / 2);
                        setMoney(-tmoney, 0);
                        tmp++;
                    }
                }
            }
        }
    }
    break;
```

저주카드 시나리오 – 사기꾼의 요구

[동작]

1. for문을 통해 현재 자신이 보유한 땅 중에 가장 비싼 땅을 찾고 그 땅을 반액에 판다.
2. 상대방에게 서버를 통해 삭제 데이터를 보내어, 똑 같은 화면이 나오게 한다.

[AppController()의 actionSecretCard()]

```
case 24: // 10만원 내고 비밀 카드 한 장 더 뽑기
    setMoney(-100000, 0);
    Platform.runLater(() -> {
        oneMore();
    });
    break;
```

저주카드 시나리오 – 축복 or 저주

[동작]

1. 현재 보유금액에서 10 만원을 차감하고 oneMore()메소드를 호출하여 비밀 카드를 한 장 더 뽑는다.

[AppController()의 actionSecretCard()]

```
case 3: // 상대땅 하나 가져오기
    takeLand(0);
    break;
```

[AppController()의 takeLand()]

```
case 0:// 상대땅 하나 가져오기
    for (PlanetData pd : planetData) {
        if (!pd.getOwner().equals(myPlayer.getId()) && !pd.getOwner().equals("X")) {
            yourLand.getItems().add(pd.getName());
        }
    }
    break;
```

```
else {
   choiceLand.setOnAction(event -> getChoice(yourLand, index));
```

```
// 비밀카드로 땅 선택시 동작 함수
private void getChoice(ChoiceBox<String> choiceBox, int index) {
    String name = choiceBox.getValue();

    if (name == null) {
        return;
    }

    for (int i = 0; i < planetData.size(); i++) {
        if (planetData.get(i).getName().equals(name)) {
            // 선택한 땅일때 동작
            if (index == 3) { // 건물 두개 지움
                if (planetData.get(i).getCount() == 1) {
                    setBuilding(i, -33, myPlayer);
                    setBuilding(i, -10, myPlayer);
                } else if (planetData.get(i).getCount() == 2) {
                    setBuilding(i, -10, myPlayer);
                    setBuilding(i, 13, myPlayer);

                }

            } else {
                // 플래그 추가 혹은 변경
                if (!planetData.get(i).getOwner().equals("X")) { // 상대땅 가져오기 일때만 기존 플래그 제거
                    ChangeLandFlag(i, myPlayer, -1, null); // 땅 가져올때 플래그들 변경 함수
                } else {
                    setFlag(i, myPlayer); // 플래그 설치 함수

                }
                // 소유자 변경
            }

        }
    }

    choiceLand.setVisible(false);
}
```

축복카드 시나리오 –

大 **축복1**

[동작]

1. takeLand()메소드에서 choice box 를 통해 상대 땅을 선택한다.
2. 선택 후, 확인을 누르면, 상대방 당의 소유자 이름은 자신으로 바뀌고, 깃발의 색깔도 자신의 것으로 바뀐다.
3. 건물이 있을 경우, getChoice 메소드 에서 건물을 바꿔준다.

[AppController()의 actionSecretCard()]

```
case 20: // 건물 두개 지을 땅 선택 (자신의 땅이여야함. 땅이 없을시 무효.)
    takeLand(3);
    break;
```

[AppController()의 takeLand()]

```
case 3:// 건물 두개 지을 땅 선택 (자신의 땅이여야함. 땅이 없을시 무효.)
    yourLand.setVisible(false);
    yourLandLabel.setVisible(false);
    for (PlanetData pd : planetData) {
        if (pd.getOwner().equals(myPlayer.getId())) {
            if (pd.getCount() <= 2) {
                myLand.getItems().add(pd.getName());
                cnt1++;
            }
        }
    }
    break;
}
} else if (index == 3) {
   if (cnt1 != 0) {
       choiceLand.setOnAction(event -> getChoice(myLand, index));
       cnt1 = 0;
   } else {
       Alert alert = new Alert(AlertType.WARNING);
       alert.setTitle("알림");
       alert.setHeaderText("카드 무효화");
       alert.setContentText("땅이 없으므로 이 카드는 무효화 됩니다.");
       alert.showAndWait().filter(response -> response == ButtonType.OK)
               .ifPresent(response -> SocketConnect.getOutMsg().println("ChangeTurn/" + turn));
       cnt1 = 0;
       cnt2 = 0;
       return;
   }
```

[AppController()의 getChoice()]

```
if (index == 3) { // 건물 두개 지음
    if (planetData.get(i).getCount() == 1) {
        setBuilding(i, -33, myPlayer);
        setBuilding(i, -10, myPlayer);
    } else if (planetData.get(i).getCount() == 2) {
        setBuilding(i, -10, myPlayer);
        setBuilding(i, 13, myPlayer);

    }
```

축복카드 시나리오 –

大 **축복2**

[동작]

1. takeLand메소드에서 자신의 땅 중 빌딩 건물 개수가 1나 이하인 땅만 choice box 목록에 보인다.
2. 땅을 선택 한 후, 확인을 누르면, 빌딩 두 개가 선택한 땅에 건설 된다.

[AppController()의 actionSecretCard()]

```
case 23: // 통행료 없이 땅 지나가기
    myPlayer.setCard(myPlayer.getCard() + 1);
    Platform.runLater(() -> {
        cardcnt1.setText(myPlayer.getCard() + "");
        cardcnt2.setText(yourPlayer.getCard() + "");
    });
    SocketConnect.getOutMsg().println("Card/" + myPlayer.getCard() + "/" + yourPlayer.getCard());
    break;
```

[상대 AppController getMsg()]

```
else if (rmsg[0].equals("Card")) {// 메세지 태그가 Card 일때 동작
    yourPlayer.setCard(Integer.parseInt(rmsg[1])); // 상대 카드 설정
    myPlayer.setCard(Integer.parseInt(rmsg[2]));// 나의 카드 설정
    Platform.runLater(() -> {
        cardcnt1.setText(myPlayer.getCard() + ""); // 나의 카드 표시
        cardcnt2.setText(yourPlayer.getCard() + "");// 상대 카드 표시
    });

}
```

축복카드 시나리오 – 무료통행 권

1. 비밀 카드를 뽑은 사용자는 무료 통행권 한장을 소유하게 된다. 화면에서는 card x 1이라고 표시된다.
2. 변경된 카드의 정보를 서버를 통해 상대방에게 전달한다.
3. 상대방은 getMsg() 메소드를 통해 비밀 카드를 뽑은 사용자의 카드 값을 변경한다.

[setMoney(int my, int your)]

```
// setMoney -> 상황별 Money를 셋팅 하는 삼수
public void setMoney(int my, int your) {
    int myMoney = myPlayer.getMoney(); // 현재 나의 머니 가져옴
    int yourMoney = yourPlayer.getMoney(); // 현재 상대의 머니 가져옴
    myMoney += my; // 머니 변경
    yourMoney += your; // 머니 변경
    myPlayer.setMoney(myMoney); // 변경된 나의 머니 설정
    yourPlayer.setMoney(yourMoney);// 변경된 상대의 머니 설정
    try {
        curMoney.setText("보유금액 : " + myPlayer.getMoney() + "원"); // 표시
    } catch (Exception e) {
        e.getMessage();
    }
    Platform.runLater(() -> {
        MyMoney.setText("보유금액 : " + myPlayer.getMoney() + "원"); // 나의 머니 표시
        YourMoney.setText("보유금액 : " + yourPlayer.getMoney() + "원"); // 상대의 머니 표시
        // 게임 상대방에게 변경된 값을 서버를 통해 전달.
        SocketConnect.getOutMsg().println("Money/" + myPlayer.getMoney() + "/" + yourPlayer.getMoney());
    });

    if (myPlayer.getMoney() < 0) { // 나의 머니가 0보다 작으면 게임 종료
        gameFinish(yourPlayer.getId()); // 게임종료 동작
        // 상대방에게도 게임종료를 서버를 통해 전달.
        SocketConnect.getOutMsg().println("GameResult/" + yourPlayer.getId() + "/" + myPlayer.getId());

    }
}
```

[gameFinish(String winner)]

```
// setMoney -> 상황별 Money를 셋팅 하는 삼수
public void setMoney(int my, int your) {
    int myMoney = myPlayer.getMoney(); // 현재 나의 머니 가져옴
    int yourMoney = yourPlayer.getMoney(); // 현재 상대의 머니 가져옴
    myMoney += my; // 머니 변경
    yourMoney += your; // 머니 변경
    myPlayer.setMoney(myMoney); // 변경된 나의 머니 설정
    yourPlayer.setMoney(yourMoney);// 변경된 상대의 머니 설정
    try {
        curMoney.setText("보유금액 : " + myPlayer.getMoney() + "원"); // 표시
    } catch (Exception e) {
        e.getMessage();
    }
    Platform.runLater(() -> {
        MyMoney.setText("보유금액 : " + myPlayer.getMoney() + "원"); // 나의 머니 표시
        YourMoney.setText("보유금액 : " + yourPlayer.getMoney() + "원"); // 상대의 머니 표시
        // 게임 상대방에게 변경된 값을 서버를 통해 전달.
        SocketConnect.getOutMsg().println("Money/" + myPlayer.getMoney() + "/" + yourPlayer.getMoney());
    });

    if (myPlayer.getMoney() < 0) { // 나의 머니가 0보다 작으면 게임 종료
        gameFinish(yourPlayer.getId()); // 게임종료 동작
        // 상대방에게도 게임종료를 서버를 통해 전달.
        SocketConnect.getOutMsg().println("GameResult/" + yourPlayer.getId() + "/" + myPlayer.getId());

    }
}
 } else if (rmsg[0].equals("GameResult")) {
     String winsql = "UPDATE member SET win = win+1 where id = '" + rmsg[1] + "'";
     String losesql = "UPDATE member SET lose = lose+1 where id = '" + rmsg[2] + "'";
     System.out.println("승리자:" + rmsg[1] + ",루저:" + rmsg[2]);
     dbWork(rmsg[0], winsql, tmsg); //승리자 승리 변경
     dbWork(rmsg[0], losesql, tmsg); //패배자 패배 변경
     msgSend(msg, yourPlayer);
     yourPlayer.yourPlayer = null;
     yourPlayer = null;
```

[Stop()]

```
@Override
public void stop() throws Exception {
    SocketConnect.getOutMsg().println("CheckMatching"); //매칭이 되었는지 확인
    super.stop();
    System.exit(0);
}
```

승리 - 정상승리

1. 돈 값이 변경 되면 setMoney 메소드가 실행된다.
2. setMoney 메소드에서 소지한 돈이 0보다 작은지 체크 하여 0보다 작으면 게임 종료 다이얼 로그를 띄우고 게임을 종료한다.
3. 서버에 게임 종료를 알리고 게임 결과를 DB에 저장한다.
4. 상대방에게도 게임의 종료를 알린다.

승리 – 비정상승리

[동작]

1. stop메소드를 통해 비정상 종료 시 서버에 데이터를 보낸다
2. 게임 결과를 전송한다.

[AppController()의 getMsg()]

```
btnGiveup.setOnAction((event) -> {
    SocketConnect.getOutMsg().println("GameResult/" + yourPlayer.getId() + "/" + myPlayer.getId());
    gameFinish(yourPlayer.getId());
});
```

[server]

```
} else if (rmsg[0].equals("GameResult")) {
    String winsql = "UPDATE member SET win = win+1 where id = '" + rmsg[1] + "'";
    String losesql = "UPDATE member SET lose = lose+1 where id = '" + rmsg[2] + "'";
    System.out.println("승리자:" + rmsg[1] + ",루저:" + rmsg[2]);
    dbWork(rmsg[0], winsql, tmsg); //승리자 승리 변경
    dbWork(rmsg[0], losesql, tmsg); //패배자 패배 변경
    msgSend(msg, yourPlayer);
    yourPlayer.yourPlayer = null;
    yourPlayer = null;
// gameFinish ->게임 종료시 동작함수
public void gameFinish(String winner) {
    Stage dialog = new Stage(StageStyle.UTILITY);
    dialog.initModality(Modality.WINDOW_MODAL);
    AnchorPane anchorPane = null;
    try {
        anchorPane = (AnchorPane) FXMLLoader.load(getClass().getResource("/fxml/fin_dialog.fxml"));
    } catch (IOException e2) {
        e2.printStackTrace();
    }
    Label FinMessage = (Label) anchorPane.lookup("#FinMessage");
    Button FinButton = (Button) anchorPane.lookup("#FinButton");
    FinMessage.setText(winner + " 승리");
    FinButton.setOnAction(e -> {
        dialog.close();
        SocketConnect.getOutMsg().println("GameFinish"); // 서버에 게임종료를 알려줌
        primaryStage.close();
    });
    Scene scene = new Scene(anchorPane);
    dialog.setScene(scene);
    dialog.setAlwaysOnTop(true);
    dialog.show();
    for (int i = 0; i < checkdialog.size(); i++) {
        checkdialog.get(i).close(); // 열려있는 다이얼 로그를 다 종료 시킴
    }
}
```

승리 – 항복

[동작]

1. 항복 버튼 클릭 시 서버에 게임 결과를 알려준다.
2. 서버에서 게임 결과를 DB에 처리하고 상대방에게 게임 종료를 알려준다.
3. gameFinish 메소드가 실행된다.
4. 서버에 게임 종료를 알려주고 게임이 종료된다.

5. 어려웠던 점 및 느낀점

어려웠던 점	**1. 소켓 통신 불안정** 컴퓨터 메모리 부족으로 인해, 클라이언트들이 동시에 과다 접속할 경우, 혹은 클라이언트 측에서 동시에 많은 양의 데이터를 서버에 송신 시 컴퓨터 과부하로 인하여 튕김 현상이 발생함. **2. 처음 접하는 자바 FX** 처음 접하는 자바 FX이므로, 개발 환경 구축 과정에 있어 어려움이 있었음. 하지만, 팀원들과 정보를 공유하며, 순탄하게 개발 환경을 구축할 수 있었음. 또한 한층 더 편해진 방법으로 사용자 인터페이스를 구현할 수 있었음. **3. 너무 많은 경우의 수** 게임에는 많은 경우의 수가 있으므로, 예측하지 못한 경우로 인하여 많은 버그가 발생함. QA를 통해 버그를 찾고, 차근차근 해결해 나감.
느낌 점	- 팀 프로젝트 수행 시, 팀원 각자가 동시에 자신이 맡은 역할을 수행하기 때문에, 코드 종합 시의 편리를 위해 누가, 어느 부분을 추가 혹은 수정하였는지 주석으로 알리는 것이 필요함. - 기능 추가 후, 왜 추가하였고, 코드에서 전반적으로 어떻게 동작하는지 팀원들에게 간략한 설명 필요.

맡은 역할	1. 회원가입 2. 비밀카드 메서드 3. 게임 설명 4. 전체 디자인 5. 버그 점검 및 수정