

「Robot Recall」 최종 보고서

1. 프로젝트 목표

1.1 JAVA 내부, 외부 라이브러리를 사용해본다.

- * JAVA 내부 Graphics2D 라이브러리 사용
- * MySQL과 eclipse 연동을 통한 Database 사용
- * JAVA nio를 이용한 네트워크 통신 구축 - 네이버 클라우드 플랫폼 서버 이용

1.2 각자 흥미 분야에 맞는 분야를 공부해본다.

- * 게임 프로그래밍 개발 프로세스 경험
- * 여러 디자인 패턴 적용
- * Database 사용환경 구축 및 프로젝트에 연동, 응용
- * 네트워크 환경 구축 및 통신

1.3 팀 프로젝트를 통한 협업 프로세스의 이해

- * Trello를 이용한 일정관리, 자료공유

2. 시스템 목표

로봇리콜?

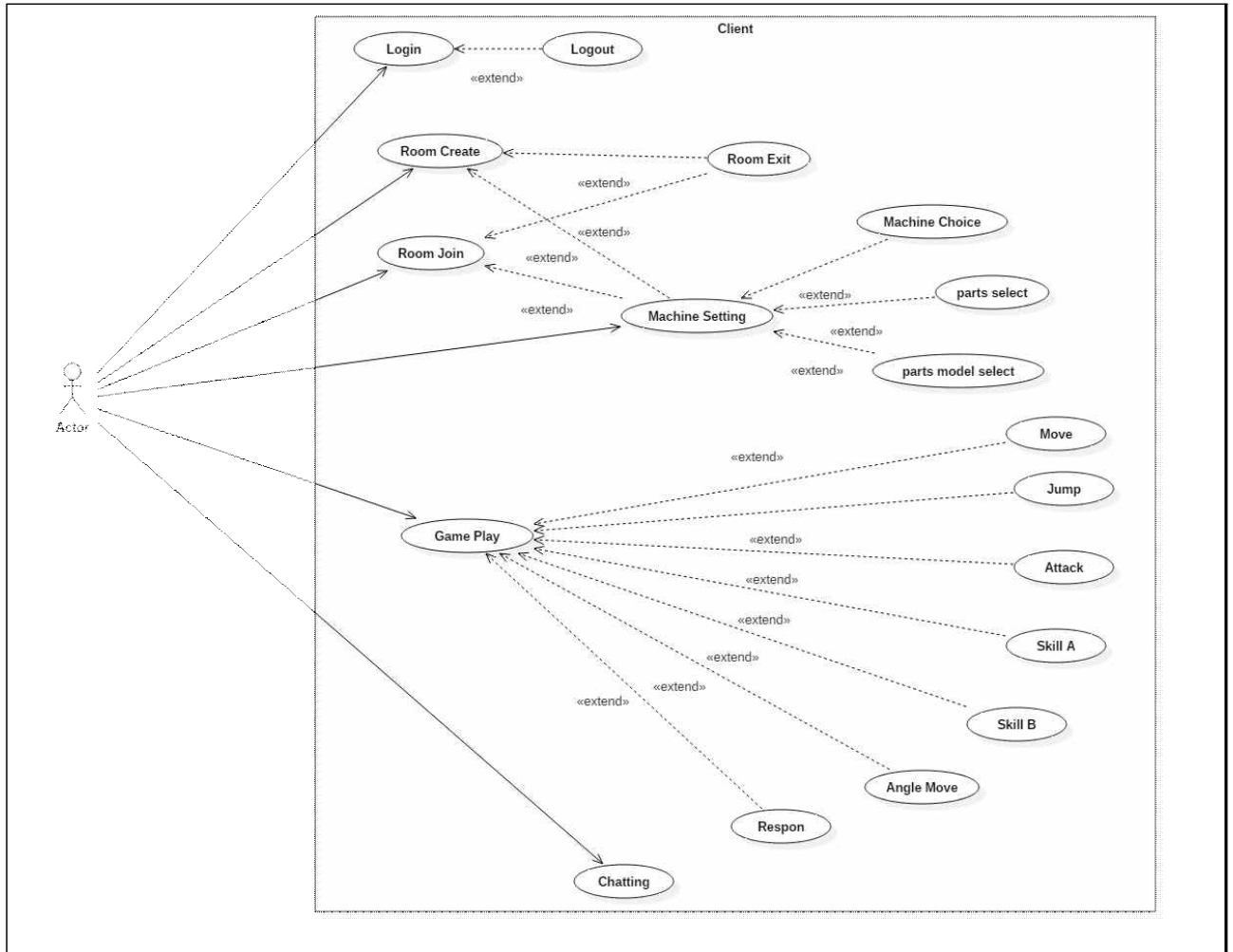
- 3개의 파츠로 구성된 로봇들이 서로 싸우는 게임
- 로봇(객체)이 총(객체)으로 총알(객체)을 쏜다.
- 총알(객체)에 맞은 로봇(객체)는 상체(객체)의 종류에 따라 데미지를 입는다.
- 상체, 하체, 무기 파츠는 각각 3종류가 존재, 각각 다른 기능을 한다.
 - * 객체간의 상호작용구현
 - * 객체간의 상속, 인터페이스 구현으로 다양한 기능을 구현

3. 요구사항 명세

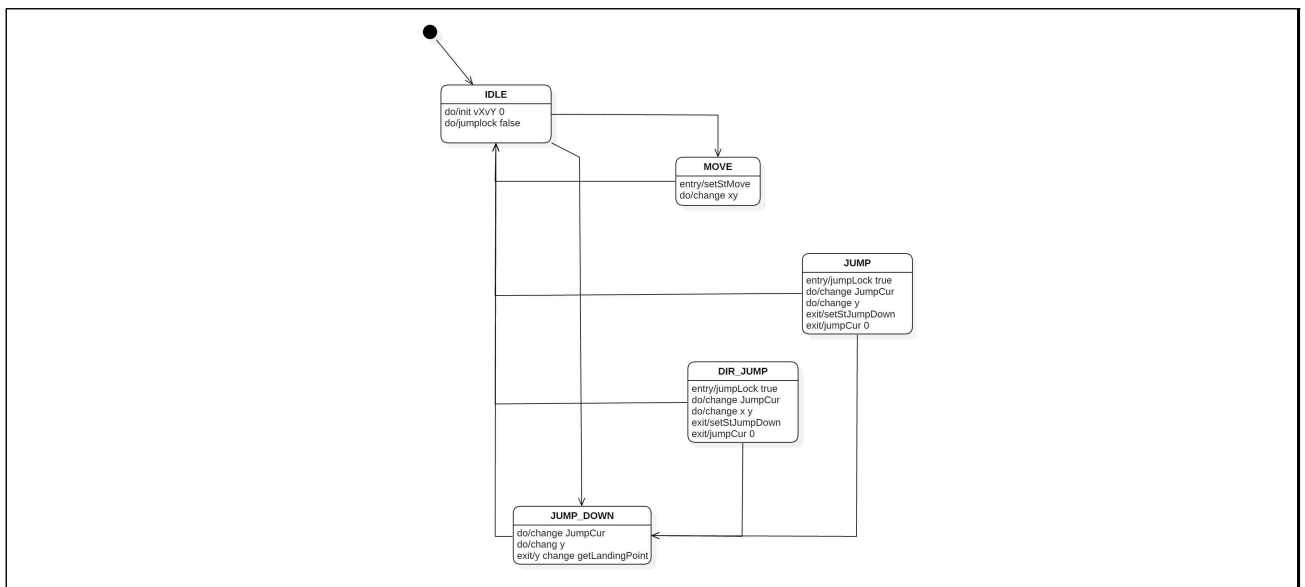
- * [붙임1] 요구사항 2.1에 자세하게 명세

4. 설계도

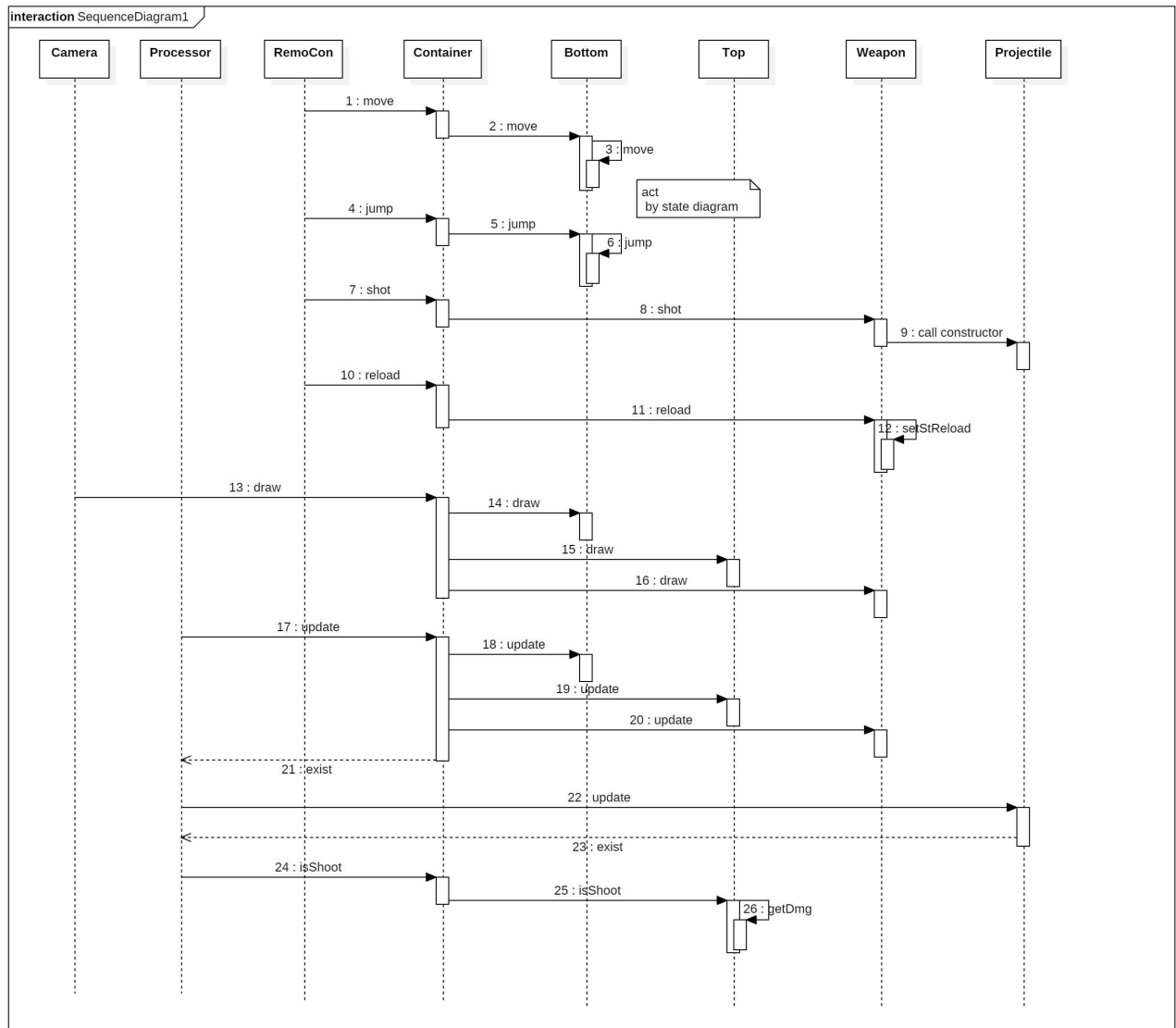
4.1 Use-Case Diagram



4.2 State Diagram



4.3 Sequence Diagram



4.4 Class Diagram

* [붙임2] 클래스 분석

5. 주요동작 테스트코드

* [붙임3] 테스트

6. 주요기능 설명 및 스크린샷

6.1 로봇의 이동기능

- 상태에 따라 행동하는 하체 파츠에 이동 및 점프 기능을 구현
- 6.3 패턴 적용사항에서 추가 설명

6.2 충돌기능

6.2.1 중력

- 객체가 떠있는지 검사하는 isFloating()

	<ul style="list-style-type: none"> - Bottom 클래스의 좌표를 기준으로 한다. - height가 1px만큼 +방향으로 큰 bound의 밑변을 기준으로 한다. - 땅을 모델링한 Land객체들과 intersects 연산을 하고, 맞으면 false, 아니면 true를 반환한다. <p>* 땅을 밟았는지 아닌지 판단할 수 있는 메소드</p> <p>* 점프입력, 이동입력을 받았을 때 판정기준이 된다.</p>
--	--

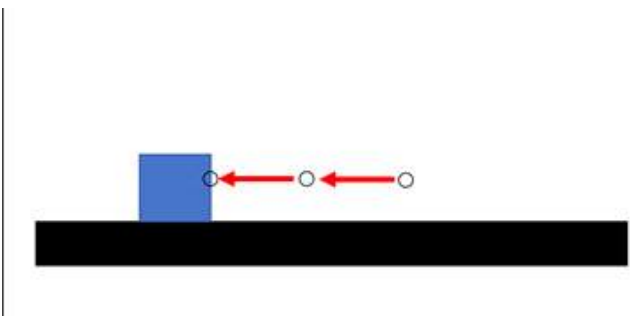
- 착지 포인트 받아오기

	<ul style="list-style-type: none"> - 파란색 Box는 객체, 하늘색 Box는 객체가 다음에 이동할 만큼의 거리를 표현한다. - 하늘색 객체의 height는 중력가속도를 받아 변화하는 y값의 증가량, vY를 의미한다. - 이 객체가 다음 update에서 이동할 vY의 bound와 Land의 리스트와 충돌 비교 iterator를 수행한다. - 다음 update시 이동할 거리중에 Land가 존재하면 Land의 가장 작은 y값(빨간 선)의 좌표를 리턴한다. - Land가 존재하지 않는 경우는 0을 리턴한다.
--	---

6.2.2 총알피격

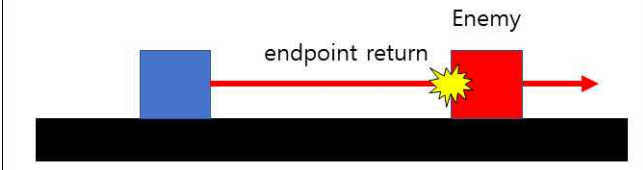
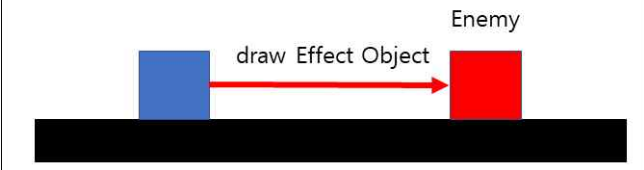
6.2.2.1 투사체 판정

- 총알 피격 검사 isShoot()

	<ul style="list-style-type: none"> - 피격당할 수 있는 객체는 모두 'Hitable' 인터페이스를 implements 한다. - 총알들은 따로 리스트에서 관리한다. - 각각 좌표들을 모두 update한 뒤, 총알은 GameObject에서 instanceof 연산으로 Hitable 인터페이스를 받은 객체에 한해서 충돌검사를 수행한다. - 총알의 bound와 객체의 bound intersect가 true이면 충돌을 감지한다. - 객체에게 영향을 주는 메소드를 호출한 뒤, 총알은 소멸한다.
---	--

6.2.2.2 HitScan 판정

- 총알 즉시 피격판정 HitScan

	
<ul style="list-style-type: none"> - Missile에 비해 총알은 날아가는 속도가 실제 세계에서든 굉장히 빠름 - 날아가면서 충돌판정을 하는 것이 아닌, 쏘는 순간 Range 내에서 객체와의 충돌을 판정한다. - 충돌이 결정 되면, Range 내의 충돌점을 반환한다. - 시작점과 끝점을 가진 Line에 Effect 객체를 소환, Draw를 한다. 	

6.3 패턴 적용 사항

6.3.1 팩토리 패턴

6.3.1.1 총알 팩토리

- 총알의 종류는 한정되어 있고, 로봇의 무기 파츠에서 생산해낸다.
- 총알의 데미지, 이동거리, 모양 등은 사전에 정의되어 있으므로 변경되지 않는다.
- 다양한 종류의 총알 Class를 정의하고, 무기의 종류에 따라 다른 총알들을 생산해내는 총알 팩토리 Class를 만든다.

6.3.1.2 로봇 팩토리

- 로봇의 파츠는 정해져 있고, 다양한 조합이 가능하다.
- 파츠는 사전에 정의된 대로만 행동한다.
- Player는 팀, 각 파츠의 번호만 넣어서 팩토리 패턴을 적용한 로봇팩토리에서 객체를 생성한다.

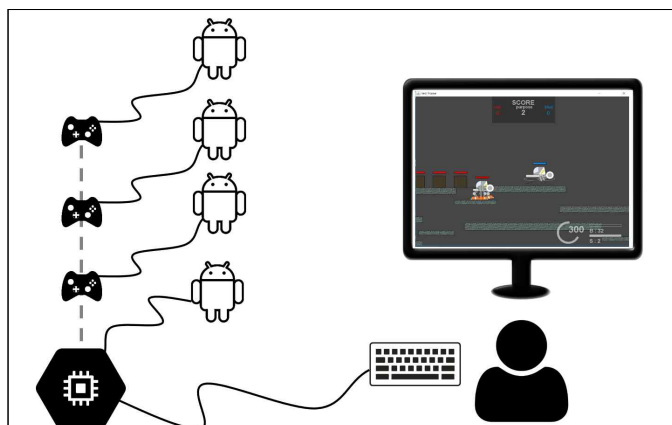
6.3.2 싱글톤 패턴

- 게임 내 Draw되는 이미지들을 File IO로 관리하는 ResourceManager
- 같은 패키지 안에 저장된 이미지들을 모두 가지고 있는 ResourceManager 클래스
- 단 한번의 인스턴스 생성으로 런타임 동안 이미지 리소스를 관리한다.
- ResourceManager 클래스를 사용하여 더욱 편하게 이미지에 접근이 가능하다.

6.3.3 FSM 패턴

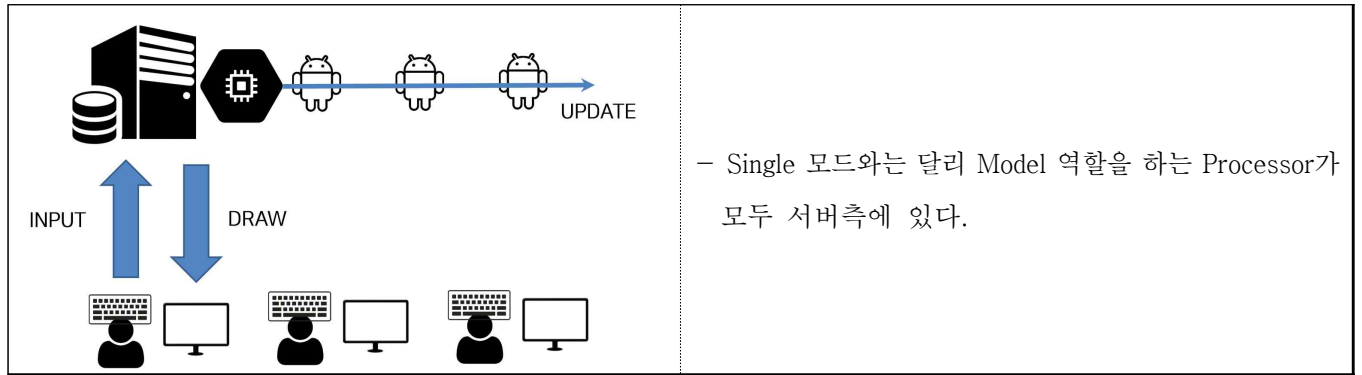
- 상태에 따라 행동하는 모델
- 4.2에서 제시한 State Diagram을 기반으로 상태에 따른 행동을 메소드로 정의
- Container 객체가 이동명령을 받아오면 하체파츠에게 이동명령을 전달

6.3.4 MVC 패턴



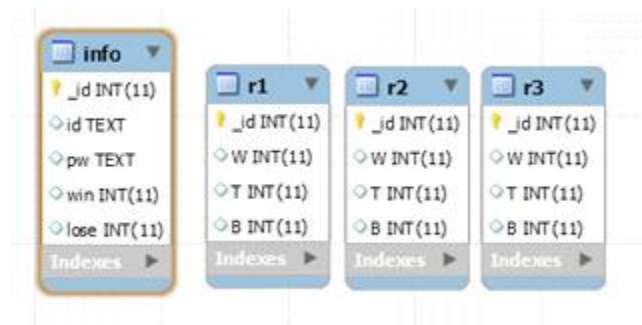
MVC 패턴 적용

1. 사용자가 키보드를 통한 입력을 한다.
2. Controller는 Model인 Robot의 상태 전이를 통해서 연산을 수행한다.
3. View는 Robot의 정보를 참조하여 Canvas에 투영한다.



6.4 Database

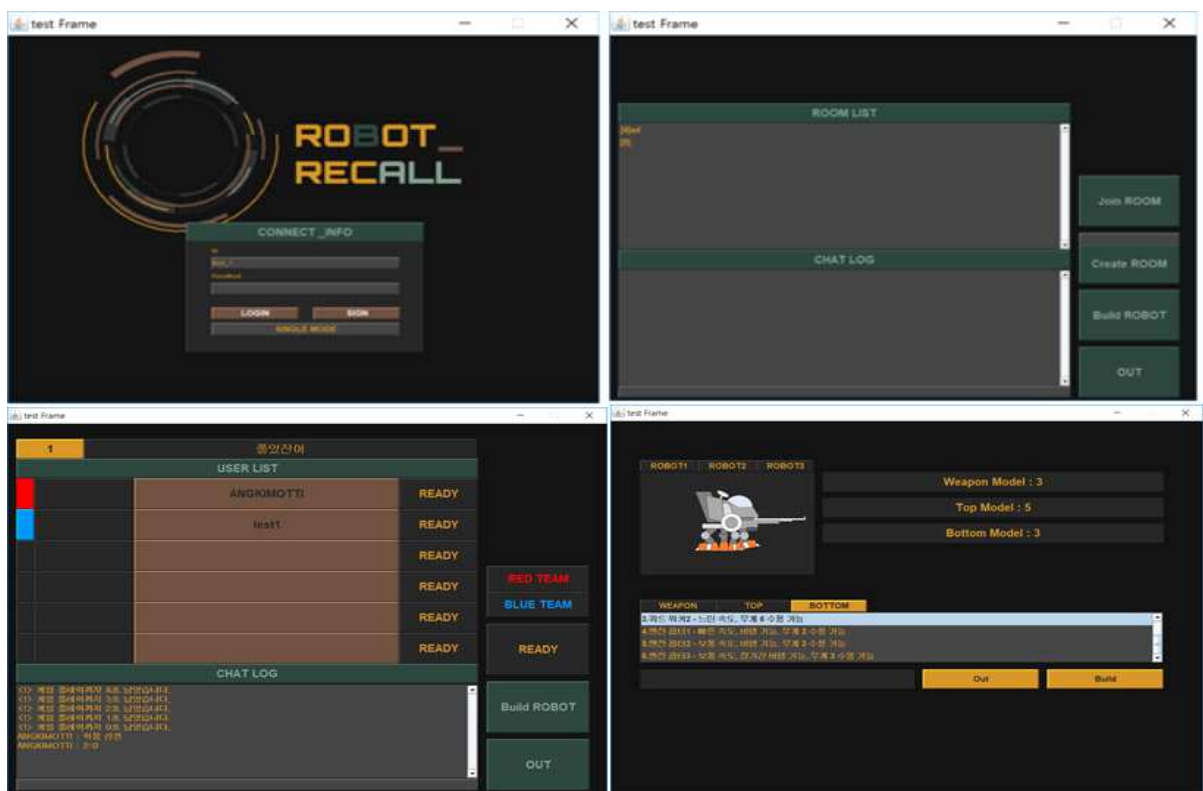
- MySQL사용, Parameter를 String인 SQL문에 연산하여 쿼리실행
- DBManager 클래스





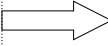
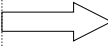
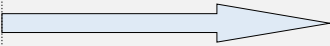
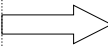
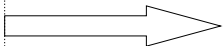

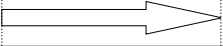
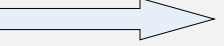

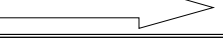
유저의 정보를 저장하기 위한 table

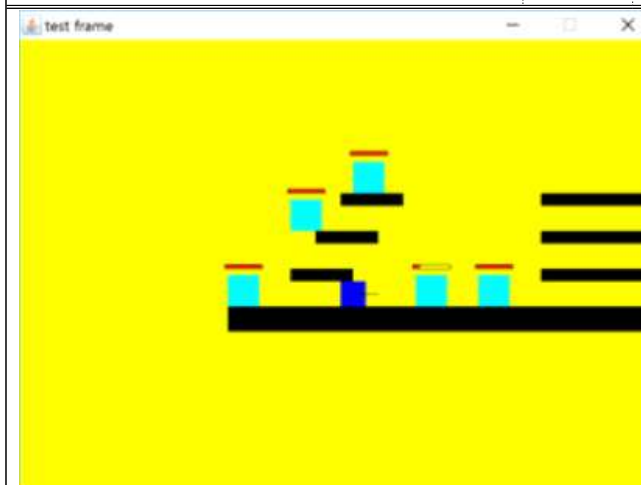
6.5 UI

- 요구사항 분석 단계에서 설계했던 것을 바탕으로 구현한 UI



6.6 개발과정

	Days	~26	~28	~29	~30	~31	~6.2	~6.4
Phase1 Base Object								
1.1 중력모델								
1.2 무기발사								
1.3 피격모델								
Phase2 Parts Detail								
2.1 파트 분화 / 특성 적용								
2.2 이미지 리소스 및 이펙트								
Phase3 Battle Situation								
3.1 적기출현(AI) 및 전투상황								
Phase4 Network Config.								
4.1 네트워크 통신을 이용한 유저간 대결 구현								



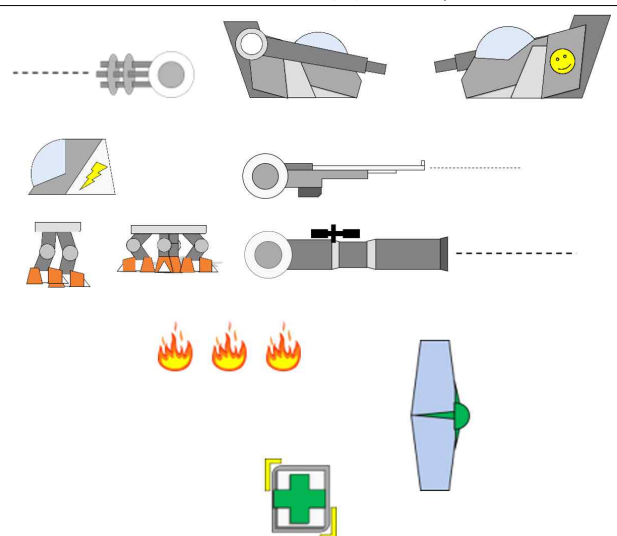
Phase 1



Phase 2 진행중간단계



Phase 4 최종 완성단계



자체 제작한 이미지 리소스들