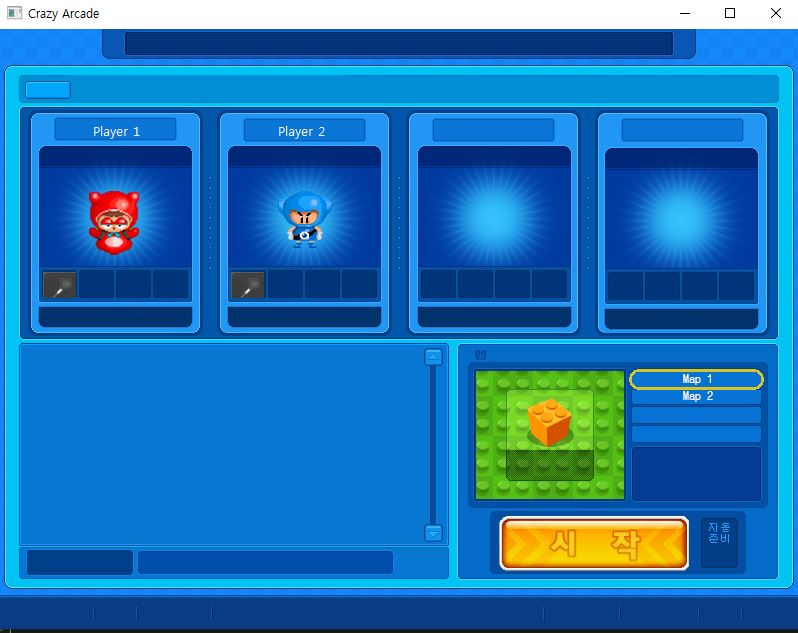
Crazy Arcade

2015182013 변진배

2016180002 고우혁

2016180045 한성재

**목차**

1. **애플리케이션 기획**
2. 개발 게임
3. 장르
4. 게임 내용
5. 게임 진행방식
6. 조작법
7. 게임 구성요소
8. **High Level**
9. 프로토콜
10. 서버, 클라이언트 흐름
11. 흐름도
12. **Low Level**
13. 패킷
14. 함수
15. **팀원 별 역할분담**
16. **개발환경**
17. **개발일정**
18. **애플리케이션 기획**
19. **개발 게임 : 크레이지 아케이드**
20. **장르**

* 2D 캐쥬얼 게임

1. **게임 내용**

* 플레이어들이 물풍선을 통해 서로 공격하여 마지막까지 살아남는 것이 목표인 배틀 로얄 형식의 게임

1. **게임 진행방식**

* 최소 3인 플레이 가능
* 플레이어들은 맵의 각 구석에 배치되며 각 플레이어는 바늘 아이템을 하나씩 가지고 시작한다.
* 플레이어들은 물풍선을 놓을 수 있으며 그 물풍선이 텨졌을 때 생기는 물줄기를 이용하여 블록들을 부수거나 다른 플레이어를 공격할 수 있다.
* 몇몇 블록안에는 아이템(이동속도 증가/물풍선 개수 증가/물풍선 파워 증가)이 있으며 블록이 부셔졌을 때 아이템을 획득할 수 있게 된다.
* 각 플레이어는 물풍선이 터졌을 때 생기는 물줄기를 맞았을 경우 물방울 속에 갇히게 된다.
* 물방울 속에 갇힌 후 일정시간이 지나거나 다른 플레이어가 부딪혀서 터트려 버릴 경우 그 플레이어는 죽게 된다.
* 물방울 속에 갇혀있을 때 바늘 아이템을 사용할 경우 한번 빠져나올 수 있으며 사용 후 바늘 아이템은 없어진다.
* 1명의 플레이어만 남을 때까지 진행한다.

1. **조작법**

* 조작방법 방향키
* 캐릭터 이동 스페이스바
* 물풍선 놓기 Ctrl
* 바늘 아이템 사용

1. **게임 구성요소**

* 800x600의 윈도우 크기
* 플레이어
* 상태 : InBubble, Escape, Die
* 능력치 : Power, Bubble, Speed
* 물풍선
* 3초 뒤 터지며 물풍선끼리 서로 연쇄 폭발
* 부서지는 타일
* 박스
* 부서지지 않는 타일
* 나무
* 집

1. **High Level**
2. **프로토콜 : TCP**

* 고정 + 가변 길이 데이터 전송 방식을 선택한다.

1. **서버, 클라이언트 흐름**
2. 서버

* 클라이언트의 접속을 기다린다. (listen)
* accept가 이루어질 때마다 각 클라이언트에 대한 thread를 만들어주고, 연결된 클라이언트들에게 접속한 클라이언트의 몇 번째 플레이어인지의 정보를 보낸다.
* 총 3번의 accept가 이루어진 이후 모든 클라이언트가 ready 되었다는 신호를 보내면 게임의 초기 시작 정보에 대한 패킷을 모든 클라이언트에게 송신한다.
* 게임이 시작된 이후 각 클라이언트에서 일어난 입력값, 상태 변화에 대한 패킷을 수신하면 서버에서 입력값과 상태 변화에 대한 처리(플레이어의 이동 및 이동에 따른 변화)를 진행한 후 모든 플레이어에게 변동된 값을 송신한다.
* 물풍선 설치에 대한 입력값을 받으면 해당 물풍선의 파워와 위치를 모든 플레이어에게 전송하고, 전송이 끝나면 시간을 계산하여 터질 때 벽과의 충돌체크를 진행한다.
* n명의 플레이어 중 n-1명의 플레이어의 상태가 dead라면 자신의 상태가 죽었는지 살았는지 여부에 따라 승리 여부를 판단하여 클라이언트들에게 결과 정보를 송신한 뒤 시작 전으로 돌아간다.

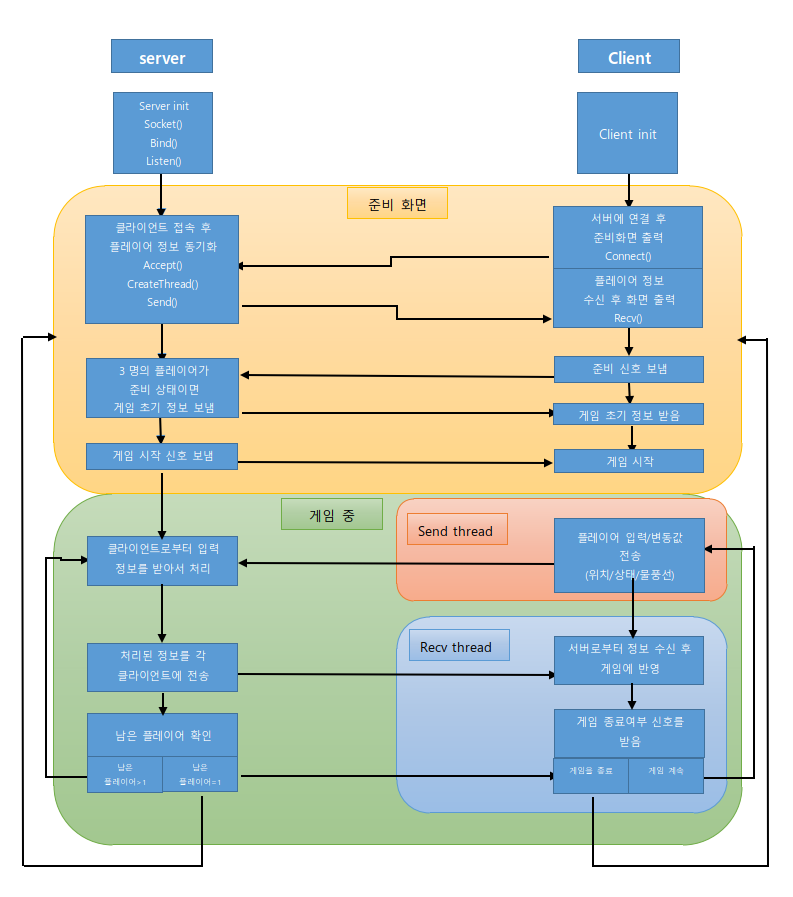
1. 클라이언트

* 서버에 연결 요청 후, 정상 접속되면 서버에서 현재 접속한 클라이언트들의 닉네임과 자신이 몇 번째 플레이어인지 정보를 수신한다.
* 이 정보를 토대로 로비 화면을 표시하고, 레디 버튼을 누르면 준비 완료에 대한 정보를 서버에 송신한다.
* 게임 초기 시작 정보에 대한 패킷(플레이어 위치, 아이템의 위치)을 수신하면 정보를 저장하고 게임을 시작한다.
* 게임이 시작된 이후 클라이언트는 키 입력을 받거나 상태에 대한 정보가 변화할 때마다 서버에 패킷을 송신한다.
* 서버로부터 패킷(플레이어의 좌표, 상태, 물풍선에 대한 정보)를 수신하면 이에 대한 처리(애니메이션, 물풍선 충돌체크)를 진행한다.
* 서버에서 게임 결과 패킷을 수신하면 결과를 표시한 뒤 게임을 끝낸다.

1. 동기화

* 서버에서 데이터를 각 플레이어에게 송신할 때, 다른 쓰레드의 송신 내용과 섞이지 않도록 이벤트를 통해 순차적으로 처리될 수 있게 설계한다.

1. **흐름도**



1. **Low Level**
2. **패킷**

* 클라이언트 패킷 : 클라이언트가 송신하는 2byte 크기의 데이터

enum ClientPacket{

left=1;

right=2;

top=4;

bottom=8;

space=16;

ctrl=32;

dead=64;

trapped=128;

}

* 패킷 : 서버에서 보낼 패킷 구조체

struct Packet{

int type;

}

* 플레이어 패킷 : 플레이어의 정보와 좌표, 상태를 담은 패킷

struct PlayerPacket{

int idx;

int x,y;

ushort status;

}

* 버블 패킷 : 물방울의 파워, 좌표를 담은 패킷

struct BubblePacket{

int power;

int x,y;

}

1. **함수**

* struct StartPacket : 플레이어의 위치와 아이템의 위치 정보를 담은 패킷
* sendPacket() : 클라이언트에서 서버로 패킷을 송신할 때 사용될 스레드 함수
* recvPacket() : 클라이언트에서 서버로부터의 패킷을 수신받는 스레드 함수
* ThreadFunc() : 서버에서 클라이언트가 접속할 때 생성할 스레드 함수

ThreadFunc()

{

recv(클라이언트가 보낸 데이터)

받은 데이터의 값에 따라 조건문으로

switch-case(이동일 경우, 물풍선을 놨을 경우, 상태가 변했을 경우에 대해)

update()

send(모든 클라언트들에게)

}

1. **팀원 별 역할분담**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 변진배 | 고우혁 | 한성재 |
| 역할 | 게임 데이터 네트워킹에 알맞게 수정 | 클라이언트 네트워크 기능 구현 | 서버 네트워크 기능 구현 |
| 서버에서 받은 데이터 처리 및 동기화 알고리즘 구현 | 클라이언트 Send / Recv 스레드 구현 | 서버 송수신 기능 구현 |

1. **개발 환경**

|  |  |
| --- | --- |
| **Operating System** | Windows 10 |
| **Integrated Development Environment** | Visual Studio 2019 Community |
| **Language** | C / C++ |
| **Application Programming Interface** | Window API |
| **Version Control System** | Git Hub |

1. **개발 일정**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **변진배** | **고우혁** | **한성재** |
| **~ 11 / 02** | 프로젝트 기획서 작성 / 수정 / 검토 | | |
| **11 / 03** | 프로젝트 기획서 제출 및 피드백 | | |
| **11 / 04** |  | 게임 코드 분석 |  |
| **11 / 05** |  |  |
| **11 / 06** |  |  |
| **11 / 07** |  |  |
| **11 / 08** |  |  |
| **11 / 09** | 기존 게임 프로젝트 함수 수정 및 보완 | 클라이언트 네트워크 기능 구현 및 네트워크 게임에 맞도록 게임 코드 수정 | 서버 프레임워크 구현 |
| **11 / 10** |
| **11 / 11** |
| **11 / 12** | 클라이언트에서 서버로 보낼 패킷 구조체 데이터 송수신에 맞게 구현  (현재 enum ClientPacket 으로 2바이트 크기로 보낼 예정) | 서버 TCP 네트워크 환경 구현 |
| **11 / 13** |
| **11 / 14** |  |
| **11 / 15** |  |  |  |
| **11 / 16** | 서버에서 클라이언트로 보낼 패킷 구조체 구현 | 클라이언트 Send, Recv 스레드 구현 | 서버 – 클라이언트 간의 데이터 송수신 구현 |
| **11 / 17** |
| **11 / 18** |  | 서버 내 멀티 Thread 구현 |
| **11 / 19** | 서버에서 recv한 데이터 처리 함수 구현 |
| **11 / 20** |  |
| **11 / 21** |  | 로비화면에서의 Send 구현 | 로비화면에서의 서버 기능 구현 |
| **11 / 22** | 클라이언트에서 recv한 데이터 처리 함수 구현 |
| **11 / 23** | 입력값에 대한 Send 구현 | 서버의 데이터 연산 / 처리 기능 구현 |
| **11 / 24** |  |
| **11 / 25** | Wait\*과 Event를 이용해 서버 멀티쓰레드 동기화 구현 | 상태에 대한 Send 구현 | 물풍선 충돌 처리 구현 |
| **11 / 26** |
| **11 / 27** | 패킷 구조체 필요할 경우 추가, 수정 및 보완 | Recv로 받은 데이터 저장 구현 및 처리 구현 | 동기화 구현 |
| **11 / 28** |
| **11 / 29** |  | 데이터 송수신 기능 수정 |
| **11 / 30** | 서버와 클라이언트의 메모리와 속도 최적화 |
| **12 / 01** | 테스트 및 오류 수정 | 테스트 및 오류 수정 |
| **12 / 02** | 서버-클라이언트 간 테스트 및 오류 수정 |
| **12 / 03** |
| **12 / 04** | 최종 점검 | 최종 점검 | 최종 점검 |
| **12 / 05** |
| **12 / 06** |
| **12 / 07** | 발표 | | |