네트워크 게임 프로그래밍

기획서 & 계획추진서

2016180022 박찬얼

2016182034 임종찬

목차

**간단한 게임 소개** **3**

**개발 환경 소개** **4**

**하이 레벨 디자인** **4,5**

게임의 흐름 Flowchart 4

네트워크 통신 Flowchart 5

**로우 레벨 디자인** **6,7,8,9,10,11**

각 요소별 기능 6

데이터 타입 정의 7

사용 함수 정의 8,9,10,11

**역할분담 및 일정표** **12,13,14,15**

구성원 역할 분담 12

역할에 따른 일정표 12,13,14,15

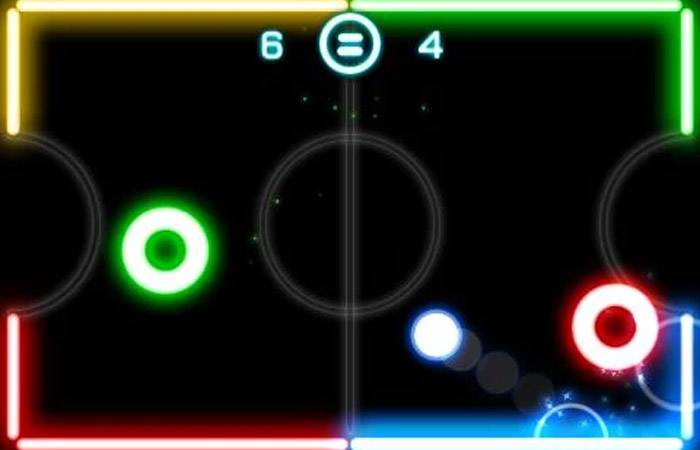
1. 간단한 게임 소개

간단한 에어 하키 게임을 구현하고 네트워크 기능을 추가하여 2인 멀티 플레이가

가능하도록 만들고자 합니다.

실시간으로 두 플레이어가 하키라켓을 움직이며, 공과 충돌하여 공을 움직이게 됩니다. 공은 라켓과 서킷에 충돌을 하면서 가속도, 각도 등의 요소를 기반으로 이동을 하게 됩니다. 최종적으로 공이 한 플레이어의 골대에 들어가게 되면 한 세트가 종료되면서 득점에 성공한 플레이어의 점수가 1점 올라가게 됩니다. 이 과정을 반복하면서 스코어가 먼저 10점에 도달한 플레이어가 승리하고 게임은 종료됩니다.

각 플레이어의 라켓과 공의 움직임에 대한 정보 등의 **실시간 네트워크 통신**과, 라켓이 공과 충돌했을 경우, 공이 서킷에 충돌했을 경우 등의 **조건부 네트워크 통신**이 모두 요구되는 게임이 예상됩니다. 예상 구현도는 아래 그림과 같습니다.

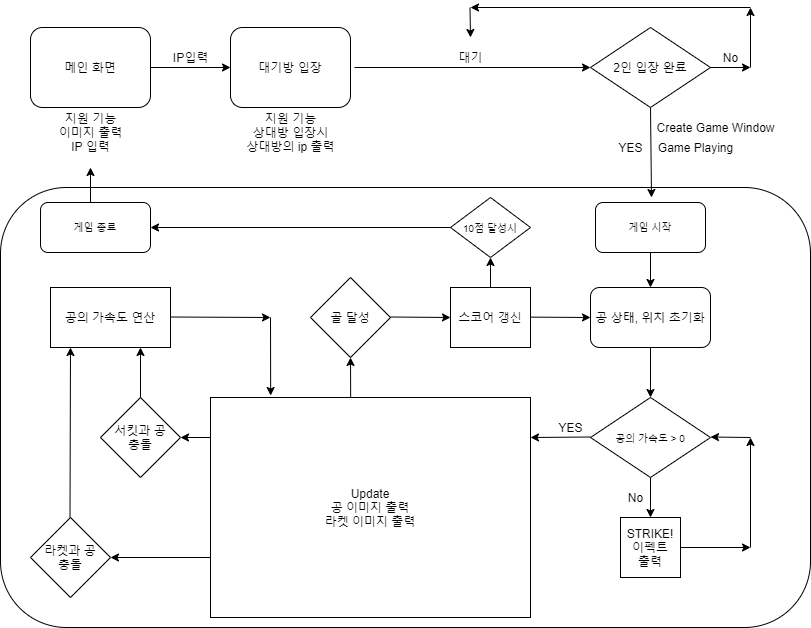


[그림- 1] 에어 하키 게임의 예상 구현도

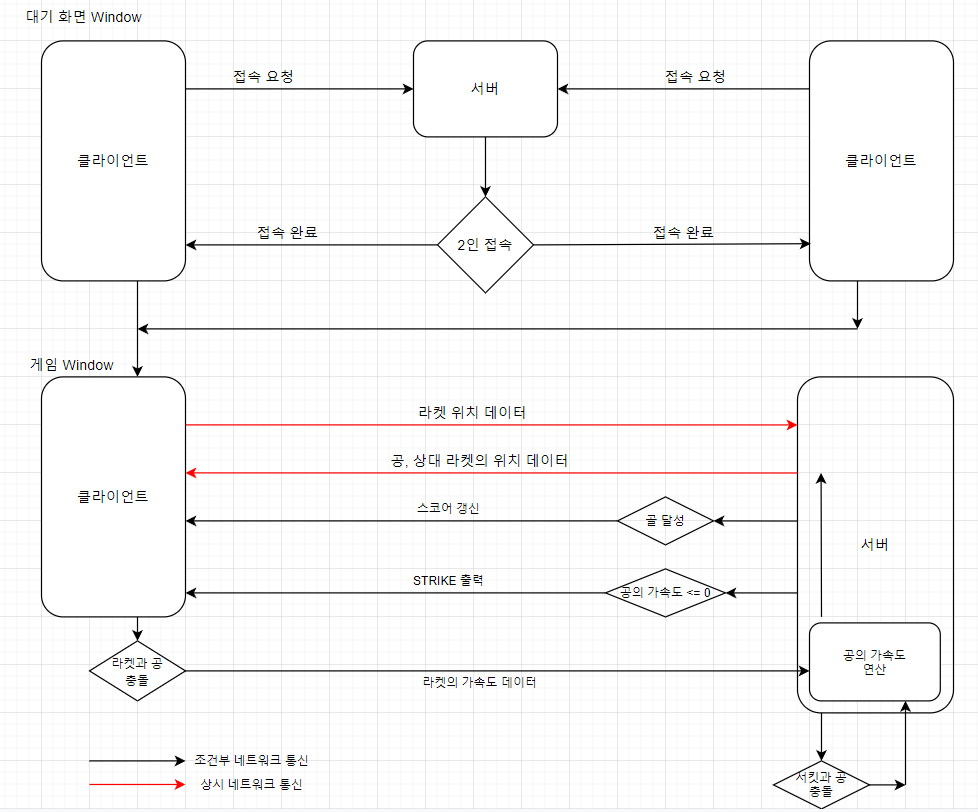
1. 개발 환경 소개

윈도우 API에 TCP 환경의 네트워크 프로토콜을 이용하여 개발을 진행할 예정이며, 게임에 사용할 이미지 리소스의 경우 자체 제작 이미지를 사용할 예정입니다.

1. 하이 레벨 디자인



[그림- 2] 게임의 흐름 Flowchart



[그림- 3] 네트워크 통신 Flowchart

1. 로우 레벨 디자인

-각 요소별 기능

1. 플레이어

플레이어 라켓의 위치정보는 마우스의 위치를 기반으로 사용합니다. 별도의 클릭 없이 마우스의 이동만으로 라켓을 움직일 수 있습니다. 라켓의 위치 정보를 실시간으로 서버와 네트워크 통신하여 주고받으며, 공과 충돌할 경우 라켓의 가속도, 충돌 각도 등에 대한 통신이 이루어집니다. 위치 정보와 가속도 정보는 각각 Point2D, Accel2D 구조체로 관리할 예정입니다.

2. 공

공은 게임 시작 시, 시작 위치에 이동됩니다. 이후 각 플레이어 라켓, 서킷과의 충돌로 서킷 내부를 이동하며, 골 라인을 넘어갔을 경우 서버에서 득점 판정이 일어납니다. 공의 위치와 가속도 정보도 플레이어와 동일한 구조체 Point2D, Accel2D로 관리할 예정입니다.

3. 서킷

서킷은 맵의 역할을 함과 동시에 공과 플레이어의 이동을 제한하는 역할을 합니다. 서킷의 고정된 크기를 바탕으로 공 또는 플레이어 라켓과의 충돌 체크에도 쓰입니다. 현재 예정중인 서킷의 크기는 800 x 400 px입니다.

-데이터 타입 정의

기본 통신 양식: 고정길이(전달할 데이터의 내용을 나타내는 헤더) + 가변길이(실제 데이터)

1. 고정길이 헤더

**~~0000 GAME\_START~~** ~~모든 사용자가 서버에 연결됨(데이터: Game\_Start)~~

플레이어 연결 방식 변경으로 인해 폐기(ver1.2)

**0001 P\_POSITION** 플레이어 라켓의 위치(데이터: pPosition)

**0002 B\_POSITION** 공의 위치(데이터: bPosition)

**0003 GOAL** 스코어 변경(데이터: score)

**0004 MOVE\_BALL** STRIKE! 이펙트 출력(데이터: no\_moveBall)

**0005 BTR\_COLLIDE** 공과 라켓의 충돌(데이터: pAccel, angleCollide)

1. 상세 데이터 타입

**~~unsigned long clientAddr[2]~~** ~~//변환된 두 사용자의 ip 주소~~

**~~struct clientInfo { SOCKET\* client1, SOCKET\* client2 }~~** ~~//양 클라이언트의 소켓 정보~~

**~~bool Game\_Start~~** ~~//두 사용자의 게임 준비 완료~~

플레이어 연결 방식 변경으로 인해 폐기(ver1.2)

**~~bool BTR\_Collide~~** ~~//공과 라켓의 충돌~~

**~~bool BTC\_Collide~~** ~~//공과 서킷의 충돌~~

**~~bool thro\_Goal~~** ~~//공이 골대를 통과했는지~~

**~~bool nomove\_Ball~~** ~~//공의 가속도 크기가 0보다 작으면~~

외부 함수 사용 폐기로 인한 삭제(ver1.3)

**bool P1Goal, P2Goal** //각 플레이어의 골 데이터

골 정보 전송 규격 변경에 따른 서버 내부 변수 추가(ver3.1)

**struct Point2D { float x\_position, y\_position }**  //객체의 서킷에서의 위치

- pPosition[2] - 플레이어 라켓의 위치

- bPosition - 공의 위치

**struct Accel2D { float x\_accel, y\_accel }**  //객체의 가속도

~~- pAccel[2] - 플레이어 라켓의 가속도~~

충돌 관련 데이터 통신으로 받아서 바로 처리하도록 변경(ver3.1)

- bAccel - 공의 가속도

**~~float angleCollide~~** ~~//공의 라켓과의 충돌 각도~~

충돌 관련 데이터 통신으로 받아서 바로 처리하도록 변경(ver3.1)

**int Score** //현재 스코어 정보(100: 게임 종료)

-사용 함수 정의

T\_는 공통 함수, S\_는 서버의 함수, C\_는 클라이언트의 함수

1. 공통 함수

**~~void T\_recvCommand()~~** ~~//헤더 데이터를 수신 후 기능에 따라 분기~~

**~~void T\_sendCommand()~~** ~~//헤더+데이터 두 차례의 데이터 전송~~

별도의 함수 없이 기능 구현으로 변경(ver1.3)

1. 서버 함수

-스레드 함수

**DWORD WINAPI getClient(LPVOID arg)** //클라이언트의 정보 recv, 서버에 적용

**DWORD WINAPI updateClient(LPVOID arg)** //서버 데이터 업데이트 및 send

\*updateClient 스레드는 getClient 스레드의 종료를 기다린 후 실행된다(이벤트 처리)

**HANDLE recvData[2], updateData[2]** //이벤트 처리용 핸들

-기능 함수

**~~void S\_recvIP()~~** ~~//받은 ip 주소를 변환하여 저장~~

**~~void S\_checkAllConnected()~~**  ~~//플레이어가 모두 연결되었다면 Game\_Start 변수를 클라이언트에게 전송~~

플레이어 연결 방식 변경으로 인해 폐기(ver1.2)

**bool S\_checkMoveBall()** //공의 가속도가 0 이상인지 체크

**void S\_checkGoal()** //공의 위치가 Goal 라인을 넘었는지 체크

**~~void S\_checkScore(int Score)~~** ~~//스코어를 확인해서 게임의 종료여부 결정~~

**void S\_resetBall()** //공 위치와 가속도를 초기화

**~~Point2D S\_newPosition( Accel2D bAccel )~~** ~~//가속도에 따른 새로운 공의 위치 계산~~

updateBall 함수에 통합 변경(ver1.4)

**~~Accel2D S\_BTCUpdate( Accel2D bAccel )~~** ~~//공과 서킷의 충돌로 인한 공의 가속도 재계산~~

**Accel2D circuitCollide(Accel2D Accel, int collideType)**

circuitCollide 함수로 이름 변경, 인자 타입 추가(ver1.5)

**~~Accel2D S\_BTRUpdate( Accel2D bAccel )~~** ~~//공과 라켓의 충돌 처리로 인한 공의 가속도 재계산~~

별도의 함수 없이 기능 구현으로 변경(ver3.1)

**void S\_dataUpdate(){** //서버에서 관리하는 데이터 업데이트

S\_checkMoveBall()

S\_checkGoal()

~~S\_BTCUpdate()~~

~~if (BTR\_Collide ) S\_BTRUpdate()~~

updateBall 에 두 기능 통합하고 추가

updateBall()

**}**

**~~void S\_sendUpdate( int clientNum ){~~** ~~//서버 데이터 전송~~

~~//조건부 업데이트~~

~~if (thro\_Goal) {~~

~~T\_sendCommand(score) //골이 날 경우, 스코어 갱신~~

~~S\_resetBall() //공의 위치와 가속도 초기화~~

~~}~~

~~if (nomove\_Ball) T\_sendCommand(nomove\_ball) //공이 움직이지 않을 경우, STRIKE! 출력을 위한 bool 변수 전송~~

check 함수에서 확인하고 이후에 바로 처리하도록 변경(ver1.3)

//상시 업데이트

T\_sendCommand(bPosition) //공의 위치 헤더+데이터 형식 전송

T\_sendCommand(pPosition[clientNum]) //상대 라켓 위치 헤더+데이터 형식 전송

**}**

**~~void S\_netUpdate(){~~**

~~for (2) S\_sendUpdate() //양 클라에게 모두 데이터 전송~~

**~~}~~**

**~~void S\_update(){~~** ~~//데이터+네트워크 업데이트~~

~~S\_dataUpdate()~~

~~S\_netUpdate()~~

**~~}~~**

**-최종 서버 진행과정 정리**

**getClient(LPVOID arg)**

**{**

***//클라이언트 연결 설정***

***//updateClient 이벤트 종료 대기***

WaitForSingleOjbect(updateData)

***//헤더 데이터 수신***

***//헤더 별 분기***

switch(header)

case P\_POSITON:

//각 플레이어의 현재 포지션 데이터 수신 후 저장

case RACKET\_COLLIDE:

//충돌 시 해당 라켓의 가속도 수신 후 공 가속도에 적용

***//이벤트 활성화***

SetEvent(recvData)

**}**

**updateClient(LPVOID arg)**

**{**

***//클라이언트 연결 설정***

***//getClient 이벤트 종료 대기***

WaitForSingleOjbect(recvData)

***//업데이트 및 헤더 할당***

updateBall(bAccel)

checkGoal()

checkMoveBall()

***//헤더 별 송신***

switch(header)

case GOAL:

//스코어 정보를 보낼 헤더와 스코어 송신

case MOVE\_BALL:

//STRIKE 이미지를 출력할 헤더와 데이터 송신

case B\_POSITION:

//상시 통신 헤더와 데이터 송신(공 위치, 상대 플레이어 위치)

***//이벤트 활성화***

SetEvent(updateData)

**}**

3. 클라이언트 함수

**void C\_printCircuit()** //서킷 이미지 출력

**void C\_pirntScore(int Score)** //스코어 출력

**void C\_printRacket(Point2D pPosition)** //라켓 이미지 출력

**void C\_printBall(Point2D bPosition)**  //공 이미지 출력

**void C\_printGoal()** //골 이미지 출력

**void C\_printStrike(Point2D bPosition)** //STRIKE! 이미지 출력

**void C\_Check\_Gameover(int Score)** //게임 종료 판정

**Point2D C\_newBallPosition (Point2D bPosition)** //새로운 공 위치 업데이트

**Point2D C\_newRacketPosition (Point2D pPosition[0]** //마우스 위치 라켓 포지션에 업데이트

**Accel2D C\_newRacketAccel (Point2D pAccel)** //라켓 가속도 업데이트

**Point2D C\_newERacketPosition (Point2D pPosition[1])** //새로운 상대 라켓 위치 업데이트(서버에서 받은 데이터 반전)

**bool C\_checkBTRCollide (Point2D bPosition, Point2D pPosition[0])** //공과 라켓의 충돌 판정

**void C\_dataUpdate(){** //클라이언트 데이터 업데이트

//조건부 업데이트

if (thro\_Goal) C\_printGoal() //골이라면, 골 이미지 출력

if (nomove\_Ball) C\_printStrike(Point2D bPosition) //공이 멈춰있다면 STRIKE! 이미지 출력

//상시 업데이트

C\_newBallPosition (Point2D bPosition) //공 위치 업데이트

C\_newERacketPosition (Point2D pPosition[1]) //상대 라켓 위치 업데이트

C\_printCircuit() //서킷 이미지 출력

C\_pirntScore() //스코어 출력

C\_printRacket(Point2D pPosition) //라켓 이미지 출력

C\_printBall(Point2D bPosition) //공 이미지 출력

**}**

**void C\_netUpdate(){** //서버에게 데이터 전송

T\_sendCommand(pPosition[0]) //라켓 위치 헤더+데이터 형식 전송

if (BTR\_Collide) {T\_sendCommand(pAccel), T\_sendCommand(angleCollide)} //만약 라켓과 공이 충돌했다면, 라켓의 가속도와 충돌 각도 전송

**}**

**void C\_update(){** //클라이언트 데이터 + 네트워크 업데이트

C\_dataUpdate()

C\_netUpdate()

**}**

1. 역할분담 및 일정표
2. 구성원 역할 분담

* 박찬얼: 게임리소스제작, 서버 영역 구현
* 임종찬: 비네트워크 기능 구현, 클라이언트 영역 구현

2. 역할에 따른 일정표

* 박찬얼

***11월***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| 31 | 1 | 2  기획서  검수 | 3 | 4 | 5 | 6  리소스제작 |
| 7  리소스제작 | 8  리소스제작 | 9 | 10 | 11 | 12 | 13  대기실 서버구현 |
| 14  서버자체  기능처리 | 15  서버자체 기능처리 | 16 | 17 | 18 | 19 | 20  네트워크  기능1  (recv) |
| 21  네트워크  기능1  (데이터가공) | 22  네트워크  기능1  (데이터가공) | 23 | 24 | 25 | 26 | 27  네트워크  기능2  (send) |
| 28  네트워크  기능2  (조건부  업데이트) | 29  네트워크  기능2  (조건부  업데이트) | 30 |  |  |  |  |

***12월***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| - | - | - | 1 | 2 | 3 | 4  클라이언트  상호작용 |
| 5  버그픽스 | 6  버그픽스 | 7  발표 | 8 | 9 | 10 | 11 |

11월 2일: 기획서 검수

6일: 리소스 제작

7일: 리소스 제작

8일: 리소스 제작

13일: S\_recvIP, S\_CheckAllConnected() 함수구현

14일: 스레드 함수 기본 구조 (getClient, updateClient) 구현

15일: S\_checkMoveBall(), S\_checkGoal(), S\_checkScore(), S\_resetBall() 함수구현

20일: recvCommand()를 사용한 분기함수 구현

21일: S\_BTCUpdate() 함수구현

22일: newPosition() 함수 구현

27일: S\_dataUpdate() 함수 구현

28일: sendCommand(bPosition,pPosition)을 이용한 상시업데이트 함수 구현

29일: S\_BTRUpdate() 함수 구현

12월 4일: 클라이언트와 상호작용하면서 정상 작동 확인.

5일: 버그픽스

6일: 버그픽스

7일: 최종 발표

-임종찬

***11월***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| 31 | 1 | 2  기획서  검수 | 3 | 4 | 5 | 6  기반 게임 개발 |
| 7  기반 게임 개발 | 8  클라이언트  소켓함수  구현 | 9 | 10 | 11 | 12 | 13  상시업뎃 함수구현 |
| 14  상시업뎃 함수구현 | 15  상시업뎃 함수구현 | 16 | 17 | 18 | 19 | 20  상시업뎃  함수구현 |
| 21  조건부  업뎃  함수구현 | 22  조건부  업뎃  함수구현 | 23 | 24 | 25 | 26 | 27  조건부  업뎃  함수구현 |
| 28  조건부  업뎃  함수구현 | 29  조건부  업뎃  함수구현 | 30 |  |  |  |  |

***12월***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| - | - | - | 1 | 2 | 3 | 4  서버 상호작용 |
| 5  버그픽스 | 6  버그픽스 | 7  발표 | 8 | 9 | 10 | 11 |

11월 2일: 기획서 검수

6일: C\_PrintCircuit(), C\_printScore(),C\_PrintRacket(),C\_PrintBall(),C\_PrintGoal() 함수구현

7일: C\_printStrike(),C\_checkGameover(),C\_newBallPosition(),C\_newRackeetPosition함수구현

8일: C\_newRacketAccel(),C\_checkBTRCollide(), 함수 구현

13일: C\_newERacketPosition, C\_dataUpdate(), 함수 구현

14일: C\_dataUpdate() 함수 구현 시작

15일: C\_dataUpdate() 함수 구현 및 개발 - 상시업데이트 함수

20일: C\_dataUpdate() 함수 구현 및 개발 - 상시업데이트 함수

21일: C\_dataUpdate() 함수 구현 및 개발 - 조건부 업데이트

22일: C\_dataUpdate() 함수 구현 및 개발 - 조건부 업데이트 이후 함수 구현 완료

27일: C\_netUpdate() 함수 구현 시작

28일: C\_netUpdate() 함수 구현 완료

29일: C\_Update() 함수 구현 완료 및 버그픽스

12월 4일: 서버와 상호작용하면서 정상 작동 확인.

5일: 버그픽스

6일: 버그픽스

7일: 최종 발표