**포 트 폴 리 오**

성 기 홍

email : [hgbb3210@naver.com](mailto:hgbb3210@naver.com)

tel : 010 - 3788 - 1344

1. **Witch of Time(시간의 마녀)**

게임 장르 : 3D 플랫포머, 샌드박스

개발 환경 : Windows 10 / Unreal Engine 4.24(클라이언트), C++(서버), Visual Studio 2019

개발 인원 : 3인

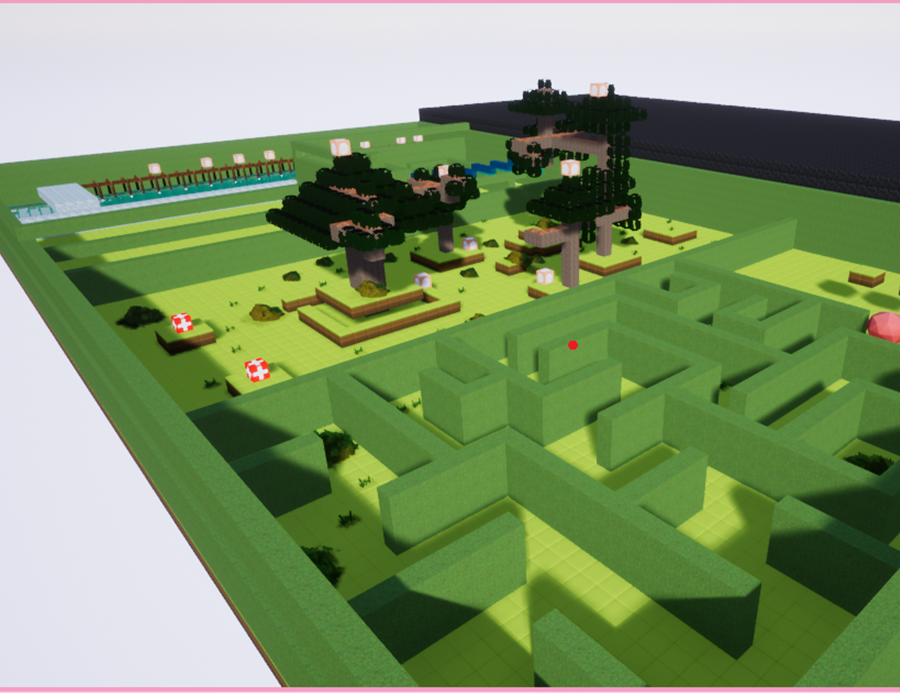
개발 기간 : 2020년 8월 ~ 2021년 9월

역할 : 서버 및 클라이언트 프로그래머

TCP/IP 서버를 IOCP서버로 변환

Unreal 4의 액터의 속성정보를 자체 프로토콜을 사용하여 C++ 서버를 통한 클라이언트 동기화 구현

**게임 스크린샷**





**게임 설명**

* ‘마인크래프트’, ‘슈퍼 마리오 메이커’ 등과 유사하게 다른 플레이어들과 함께 맵을 제작하고, 제작한 맵을 함께 플레이 할 수 있는 플랫포머 게임.

**구현 내용**

* 멀티스레드 IOCP로 제작된 서버에서 모든 게임 컨텐츠가 실행되며 언리얼 클라이언트의 동기화도 수행
* 같은 세션에서 여러 명의 플레이어가 동시에 한 맵을 제작할 수 있도록 동기화
* 기존 블록에 여러 종류의 커맨드들을 적용하여 커맨드 블록 제작 가능

**구현하며 어려웠던 점**

- 맵 로드 등 서버에 큰 부하가 걸리는 상황에서의 성능 향상

- 서버에서 클라이언트의 모든 컨텐츠를 실행

**극복 방법**

- 패킷에 최대한 많은 정보를 담아서 send / recv 호출 횟수의 최적화

- 한번에 모든 컨텐츠를 서버에서 구현하지 않고, 일단 클라이언트에 구현해서 테스트 후, 단계적으로 서버로 옮기는 방식으로 구현

**핵심 소스코드**

1. 클라이언트 - 언리얼 C++ 코드

|  |
| --- |
| void ALevelEditorPawn::DestroyBlock() {  FVector CLocation = this->GetActorLocation();  FVector CForwardVector = this->GetActorForwardVector();  FHitResult hitResult;  FCollisionQueryParams collisionParams;  collisionParams.AddIgnoredActor(this);  collisionParams.AddIgnoredActor(DummyBlock);   if (GetWorld()->LineTraceSingleByChannel(hitResult, CLocation, CLocation + CForwardVector \* 2000, ECC\_Visibility, collisionParams) && hitResult.GetActor()->ActorHasTag("Destroyable"))  {  ToDestroyBlock = hitResult.GetActor();  ToDestroyBlockName = hitResult.GetActor()->GetName();  client->send\_destroy\_packet(Cast<ABlockBase>(ToDestroyBlock)->block\_id);  } } |

* 에디터 모드의 플레이어 액터의 블럭 삭제 함수의 예
* 클라이언트의 동기화가 필요한 액터들은 서버와의 통신을 담당하는 액터의 포인터를 가짐
* 서버 동기화가 필요한 이벤트가 발생할 경우, 해당 액터에서는 서버와의 통신을 담당하는 액터의 포인터를 통하여 서버에 패킷을 전송함

1. 서버 코드 예시 - 몬스터 이동

|  |
| --- |
| for (auto monster : monster\_block\_id) {  if (monster.second == 75) // 바이킹 {  TracePacket packet;  packet.block\_id = monster.first;  float mindistance = 999999;  int minid = -1;  for (int i = 1; i <= MAX\_USER; i++) {  if (objects[i].object\_state != STATE\_INGAME) continue;  float x = powf((objects[i].x - objects[monster.first].x), 2);  float y = powf((objects[i].y - objects[monster.first].y), 2);  float z = powf((objects[i].z - objects[monster.first].z), 2);  float temp = sqrtf(x + y + z);  if (mindistance > temp)  mindistance = temp;  minid = i;  }  if (minid != -1 && mindistance < 1000) {  packet.player\_id = minid;  Broadcast\_Packet(&packet);  }   }  } |

* 기능 : 주위에 가장 가까운 플레이어에게로 이동한다.
* 서버에서 액터의 정보를 바꿀 필요가 있을 경우 서버 내의 오브젝트 정보를 수정한 후, 바뀐 오브젝트의 정보를 패킷에 넣어서 전송
* 패킷에는 수정된 오브젝트의 id값과 그 정보가 들어감

1. 언리얼 C++ 코드 - 서버 패킷과 언리얼 블루프린트 함수(move\_monster) 연동

|  |
| --- |
| case TRACE:  {  auto cast = reinterpret\_cast<TracePacket\*>(p\_buf);  move\_monster(cast->block\_id, cast->player\_id);  }  break; |

* 클라이언트가 패킷을 받으면 패킷의 정보를 매개변수로 하는 블루프린트 함수를 호출해 언리얼 액터의 property를 변경함

1. 언리얼 C++코드 - 커맨드 블럭의 자료구조

|  |
| --- |
| USTRUCT(BlueprintType) struct FCommandBlockInfo {  GENERATED\_USTRUCT\_BODY()  UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Basic)  int index;  UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Basic)  TArray<int32> data; }; |

* 모든 블럭은 커맨드 블럭의 인덱스(예 : 이동 커맨드)와 그 정보(예 : 이동 커맨드의 x좌표)를 멤버로 갖는 구조체의 TArray를 가짐
* 플레이어가 블럭에 커맨드를 적용하면 해당 자료구조에 정보가 저장됨

1. 언리얼 C++코드 - 커맨드 블럭 액터들의 상태를 서버에 전송하는 코드

|  |
| --- |
| void Aclient::send\_command\_packet(int block\_id, TArray<int32> commandindex, TArray<int32> commanddata\_0, TArray<int32> commanddata\_1, TArray<int32> commanddata\_2, TArray<int32> commanddata\_3) {  CommandPacket commandpacket;  commandpacket.commandblock\_id = block\_id;  for (int i = 0; i < lengthofcommandlist; ++i)  {  if (commandindex[i] >= 0)  {  commandpacket.commandblockindex[i] = commandindex[i];  commandpacket.commandblockdata\_0[i] = commanddata\_0[i];  commandpacket.commandblockdata\_1[i] = commanddata\_1[i];  commandpacket.commandblockdata\_2[i] = commanddata\_2[i];  commandpacket.commandblockdata\_3[i] = commanddata\_3[i];  }  }  send\_packet(&commandpacket); } |

5-2) 서버에서 받은 패킷의 정보를 다른 플레이어들에게 브로드캐스팅하는 코드

|  |
| --- |
| case COMMAND:  {  auto cast = reinterpret\_cast<CommandPacket\*>(buffer);  CommandPacket commandpacket;  commandpacket.commandblock\_id = cast->commandblock\_id;  for (int i = 0; i < COMMANDS; ++i)  {  if (cast->commandblockindex[i] >= 0)   {  commandpacket.commandblockindex[i] = cast->commandblockindex[i];  commandpacket.commandblockdata\_0[i] = cast->commandblockdata\_0[i];  commandpacket.commandblockdata\_1[i] = cast->commandblockdata\_1[i];  commandpacket.commandblockdata\_2[i] = cast->commandblockdata\_2[i];  commandpacket.commandblockdata\_3[i] = cast->commandblockdata\_3[i];  }  }  Broadcast\_Packet(&commandpacket);  } |

* 저장된 자료구조에서 정보를 읽어서 배열에 넣은 후 이를 패킷으로 만들어서 서버에 전송함
* 서버는 받은 커맨드 블럭 정보를 오브젝트에 저장한 후 다른 플레이어에게 브로드캐스팅함

영상 링크 : <https://youtu.be/6SCpTso8YUc>

소스코드 링크 :

클라이언트 - <https://github.com/yimgunho/Witch_of_Time/tree/master/Witch_of_Time/Source/Witch_of_Time>

서버 -

<https://github.com/yimgunho/Witch_of_Time/tree/master/Server/Server>

1. **MMO Simulator**

게임 장르 : MMORPG

개발 환경 : Windows 10 / C++ , Visual Studio 2019, 클라이언트 - WINAPI

개발 인원 : 1인

개발 기간 : 2021년 5월 – 2021년 7월

**게임 스크린샷**



**게임 설명**

이동하면서 만나는 몬스터와 전투하며 승리 후 얻은 경험치로 레벨업 하는 게임

**구현 내용**

* 가장 간단한 MMORPG의 구현
* 서버에 다양한 기능을 구현해 보고, 동시접속자 수 테스트

(**가상 클라이언트 프로그램으로 6000명 까지 실제 측정**)

* 멀티 스레드 IOCP 서버를 통한 모든 컨텐츠 실행
* 미리 생성된 맵 파일을 읽어서 맵 생성
* 데이터베이스 연동을 통한 사용자 데이터 관리
* 채팅 패킷을 통한 채팅 및 시스템 메시지 처리
* lua 스크립트를 통한 npc ai 적용
* 시야 처리 및 섹터 분할을 통한 이동 최적화
* 타이머 큐를 통한 몬스터의 이동 / 공격 등의 이벤트 처리
* A\* 알고리즘을 사용한 npc의 길찾기

**구현하며 어려웠던 점**

* 멀티스레드 환경에서의 Data Race 문제 해결
* 최적화를 통한 동시 접속자 수 극대화

**극복 방법**

* 데이터 레이스가 적은 자료구조(Array) 사용
* mutex를 사용한 상호 배제
* 많은 npc의 길찾기 알고리즘에서 시간을 많이 사용하지 않도록 일정 깊이까지만 경로를 탐색하도록 함
* 월드를 작은 세션으로 쪼갠 후, 같은 세션에 있는 플레이어 / npc끼리만 시야 처리

**핵심 소스코드**

1. 서버 코드 : Monster 생성 시 AI를 위한 루아 가상머신 추가

|  |
| --- |
| for (int i = 0; i < MAX\_USER + 1; i++) {  // npc 생성  auto& pl = objects[i];  pl.id = i;  pl.object\_state = STATE\_READY;  if (true == is\_npc(i)) {  {//npc의 정보를 넣는 부분, 생략}  }  sectors[pl.sec\_x][pl.sec\_y].insert(pl.id);   // npc 객체에 lua 가상 머신 생성  lua\_State\* L\_state = pl.L = luaL\_newstate();  luaL\_openlibs(L\_state);  luaL\_loadfile(L\_state, "npc.lua");  int ret = lua\_pcall(L\_state, 0, 0, 0);  // lua 함수 호출  lua\_getglobal(L\_state, "set\_object\_info");  lua\_pushnumber(L\_state, i);  lua\_pcall(L\_state, 1, 0, 0);  // c++ 함수 등록  lua\_register(L\_state, "API\_get\_x", API\_get\_x);  lua\_register(L\_state, "API\_get\_y", API\_get\_y);  lua\_register(L\_state, "API\_ATTACK\_PLAYER", API\_ATTACK\_PLAYER); } |

* 서버 시작시 npc 객체의 생성 & npc 객체에 lua 가상 머신 생성
* 가상 머신에 c++로 작성된 호출 가능한 함수 등록

1. 데이터 베이스 연동 코드 : 플레이어 정보 로드, Stored procedure로 구현

|  |
| --- |
| void db\_func() { // 변수 선언부 및 기본 코드 생략  // DB에 연결  retcode = SQLConnect(hdbc, (SQLWCHAR\*)L"my\_server", SQL\_NTS, (SQLWCHAR\*)NULL, 0, NULL, 0); while (true) {  db\_lock.lock();  // 큐에 db 접근 이벤트가 존재하는지 검사 if (!db\_queue.empty() &&  db\_queue.front().event\_time <=system\_clock::now()) {  DB\_EVENT ev = db\_queue.front();  db\_queue.pop(); // db 이벤트에 따른 처리  switch (ev.ev\_type) {  case OP\_GET\_PLAYER\_INFO:  {  retcode = SQLAllocHandle(SQL\_HANDLE\_STMT, hdbc, &hstmt);  SQLWCHAR buf[255];  wchar\_t oid[20];  MultiByteToWideChar(CP\_ACP, 0, ev.object\_id, 20, oid, 20);  // Stored Procedure 호출  wsprintf(buf, L"EXEC READ\_PLAYERINFO %s", oid);  retcode = SQLExecDirect(hstmt, (SQLWCHAR\*)buf, SQL\_NTS);  // db 이벤트 처리 완료를 알리는 이벤트 생성  OVERLAPPED\_EXTENDED\* overlapped = new OVERLAPPED\_EXTENDED;  overlapped->op\_type = ev.ev\_type;  if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO) { // Bind columns 1, 2, and 3   retcode = SQLBindCol(hstmt, 2, SQL\_C\_ULONG, &char\_level, 10, &cbCharLevel);  retcode = SQLBindCol(hstmt, 3, SQL\_C\_ULONG, &char\_hp, 10, &cbCharhp);  retcode = SQLBindCol(hstmt, 4, SQL\_C\_ULONG, &char\_exp, 10, &cbCharexp);  retcode = SQLBindCol(hstmt, 5, SQL\_C\_SHORT, &char\_x, 10, &cbCharX);  retcode = SQLBindCol(hstmt, 6, SQL\_C\_SHORT, &char\_y, 10, &cbCharY);  for (int i = 0; ; i++) {  retcode = SQLFetch(hstmt);  if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO)  {  ev.object\_exp = char\_exp;  ev.object\_hp = char\_hp;  ev.object\_level = char\_level;  ev.object\_x = char\_x;  ev.object\_y = char\_y;  }  else  {  if (i != 0)  break; // end of data  else  {  ev.object\_exp = 0;  ev.object\_hp = 100;  ev.object\_level = 1;  ev.object\_x = 1000;  ev.object\_y = 1000;  break;  }  }  }  } // Process data   SQLCancel(hstmt);  SQLFreeHandle(SQL\_HANDLE\_STMT, hstmt); // 정보를 넣어서 iocp를 통해 처리  memcpy(&overlapped->packetbuf[0], &ev, sizeof(DB\_EVENT));  PostQueuedCompletionStatus(iocp\_handle, 1, ev.object\_index, &overlapped->overlapped);  }  break; // 다른 case문 생략  db\_lock.unlock();  } else {  // 등록된 이벤트가 없음 db\_lock.unlock(); this\_thread::sleep\_for(10ms);  }  SQLDisconnect(hdbc);  SQLFreeHandle(SQL\_HANDLE\_DBC, hdbc);  SQLFreeHandle(SQL\_HANDLE\_ENV, henv);  } |
|  |

* <https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/sqlbindcol-function?redirectedfrom=MSDN&view=sql-server-ver15>
* MS의 code example을 수정해서 사용
* db 스레드에서 db 이벤트를 전담하여 처리함
* 객체에서 db에 접근할 일이 있을 경우 db 이벤트 큐에 이벤트를 push
* db 스레드에서는 이벤트 큐를 검사하여 db에 접근 후 결과를 iocp 이벤트로 처리

1. 서버코드 : 성능향상을 위한 객체 검색 최정화

|  |
| --- |
| void search\_sector(int x, int y, unordered\_set <int>& new\_vl, int p\_id) { if (x < 0 || y < 0 || x > 80 || y > 80)  return; if (is\_npc(p\_id) == true) {  for (auto& pl : sectors[x][y]) {  if (pl == p\_id) continue;  if (is\_npc(pl) == true) continue;  if (objects[pl].object\_state == STATE\_INGAME && can\_see(p\_id, pl)) new\_vl.insert(pl);  } } else {  for (auto& pl : sectors[x][y]) {  if (pl == p\_id) continue;  if (objects[pl].object\_state == STATE\_INGAME && can\_see(p\_id, pl)) new\_vl.insert(pl);  } }  } |

* 이동 이벤트 등을 처리할 필요가 있을 때, 서버의 부하를 줄이기 위해서 같은 섹터에 존재하는 플레이어들끼리 시야 처리
* 각종 이벤트는 시야 내에 있는 플레이어에게만 전송됨

1. 서버코드 : A\* 길찾기에 실행시간 제한 추가

|  |
| --- |
| int a\_star\_search(int p\_id, int o\_id) {  … // 선언부 생략  while (!open.empty()) {  open.sort(comp);  cur = open.front();  if (cur->g > 10) { // 깊이가 10이상이면 검색 종료  int dir = cur->dir;  for (auto& c : open) free\_list.recycle(c)};  for (auto& c : closed) free\_list.recycle(c)};;  return dir;  }  if (get\_dist(cur->x, cur->y, tx, ty) <= 1) {  node\* ptr = cur;  if (ptr->parent == nullptr) return -2;  while (ptr->parent->parent != nullptr) ptr = ptr->parent;  int dir = ptr->dir;  for (auto& c : open) free\_list.recycle(c)};  for (auto& c : closed) free\_list.recycle(c)};  return dir;  }  else {  if (cur->y > 0)  {  if (!obstacles[cur->x][cur->y - 1]) {  bool found = false;  child = free\_list.take();  child.set(cur->x, cur->y - 1, 0, cur); child->f = get\_dist(child->x, child->y, tx, ty) + child->g;  for (auto& c : open) {  if (c->x == child->x && c->y == child->y)  found = true;  }  for (auto& c : closed) {  if (c->x == child->x && c->y == child->y) found = true;  }  if (!found) open.push\_back(child);  else free\_list.recycle(child)};  } } // 다른 세 방향에서도 동일하게 처리, 생략}  closed.push\_back(cur);  }  open.pop\_front();  }  for (auto& c : open) {  free\_list.recycle(c)};  }  for (auto& c : closed) {  free\_list.recycle(c)};  }  return -2; } |
|  |

* A\* 알고리즘을 사용한 npc의 길찾기
* 트리의 깊이를 10으로 제한하고 탐색 실패시 최적의 결과를 리턴하여서 최적화

영상 링크 : <https://youtu.be/x_r8AD9l3og>

소스코드 링크 :

클라이언트 -

<https://github.com/SuniAnice/portfolio/tree/main/simple_mmo/Client/GSP_Client_2016182019>

서버 -

<https://github.com/SuniAnice/portfolio/tree/main/simple_mmo/Server/IOCP_Server>

1. **magic quest**

게임 장르 : 퍼즐

개발 환경 : Android / Java, Android Studio

개발 인원 : 1인

개발 기간 : 2021년 6월 – 2021년 7월

**게임 스크린샷**







**게임 설명**

* 애니팡과 같은 3-매치 퍼즐 게임에 RPG 요소를 더한 게임
* 색깔 블럭을 파괴해 마나를 모아서 스킬을 사용하고, 검 블럭을 파괴해서 적을 공격하고, 방패 블럭을 파괴하여 적의 공격을 막을 수 있음
* 적을 처치하면 랜덤으로 선택되는 보상 중에서 하나를 선택할 수 있음
* 점점 강해지는 적들을 상대로 최대한 적을 많이 처치하는 것이 목표인 로그라이크 게임

**구현 내용**

* ArrayList를 사용한 게임 루프 구현
* Scene의 Stack을 사용한 멀티플 씬
* 다양한 비트맵 클래스를 활용한 화면 그리기 / 충돌 처리

**구현하며 어려웠던 점**

* 가비지 컬렉션으로 인한 랙 발생 우려

**극복 방법**

* 게임에서 삭제된 객체의 리소스 재활용

영상 링크 : <https://youtu.be/gMUeidO9O4Q>

소스코드 링크 :

<https://github.com/SuniAnice/portfolio/tree/main/magic_quest/Project/app/src/main/java/kr/ac/kpu/game/s2016182019/termproject>

**4. Cube Beat**

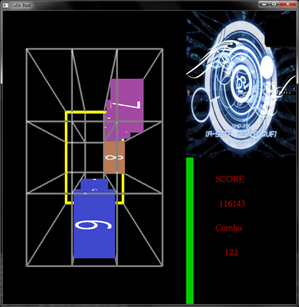
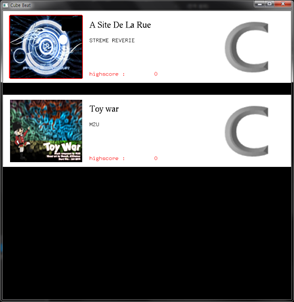
게임 장르 : 리듬 액션

개발 환경 : Windows 10 / Opengl , Visual Studio 2017

개발 인원 : 1인

개발 기간 : 2017년 12월

**게임 스크린샷**



**게임 설명**

* 3x3의 큐브 공간으로 떨어지는 노트를 올바른 타이밍에 처리하는 게임
* 특정 타이밍에 카메라를 회전시켜서 큐브 공간을 다른 방향에서 보여줌으로써 여러 느낌을 줌

**구현 내용**

* 분할된 뷰포트 중 게임 뷰포트인 좌측 뷰포트의 카메라를 회전시키면서 새로운 뷰를 모여줌
* 원근 투영과 평행 투영의 동적 전환
* Json 파일 입출력을 통한 음악의 추가 및 하이스코어 저장 구현
* FMOD를 통한 음악 재생

**핵심 소스코드**

|  |
| --- |
| void initiailze() {  srand(time(NULL)); // json 파일 입출력  ifstream notedata("../Resource/notedata.json", ifstream::binary);  notedata >> root;  ifstream high("../Resource/highscore.json", ifstream::binary);  high >> Highscore;  // fmod 시스템에 json 파일에서 읽은 경로를 가지고 사운드 생성  System\_Create(&soundSystem);  soundSystem->init(2, FMOD\_INIT\_NORMAL, NULL);  soundSystem->createSound("../Resource/title.mp3", FMOD\_LOOP\_OFF, 0, &title);  soundSystem->createSound("../Resource/gameover.mp3", FMOD\_LOOP\_OFF, 0, &resultbgm[0]);  soundSystem->createSound("../Resource/result.mp3", FMOD\_LOOP\_OFF, 0, &resultbgm[1]);  for (int i = 0; i < root["musicnum"].asInt();i++)  {  soundSystem->createSound(root["Music"][i]["Filepath"].asCString(),  FMOD\_LOOP\_OFF, 0, &sounds[i]);  } } |

* json 파일에서 읽은 정보를 자료구조로 저장, json의 파일 입출력에는 jsoncpp 라이브러리를 사용

|  |
| --- |
| void rotatetimer(int value) {  // 원근 투영과 평행 투영 전환  if (roorder == 0 || roorder == 1 || roorder == 3 || roorder == 4 || roorder == 7 || roorder == 9 || roorder == 10)  pers = 1;  else  pers = 0;  // json 파일에서 읽은 회전 정보를 바탕으로 뷰포트 회전  switch(root["Rotateorder"][roorder].asInt())  {  case 0:  glutTimerFunc(100, RotateX, 0);  break;  case 1:  glutTimerFunc(100, RotateY, 0);  break;  case 2:  glutTimerFunc(100, RotateZ, 0);  break;  }  roorder++;  // 일정 시간 후 다시 함수 호출  if (roorder <12 && state == 2)  glutTimerFunc(10000, rotatetimer, 0); } |
|  |

* json 파일에 작성된 회전 정보를 바탕으로 뷰포트를 회전함

영상 링크 : <https://youtu.be/Ce2piJ8uEBk>

소스코드 링크 :

<https://github.com/SuniAnice/portfolio/tree/main/Cube_Beat/Cube_Beat>