2D 포트폴리오

Forager 모작

김도영

제작 정보

Forager

장르 : 오픈월드 생존 크래프트

사용 툴 : WinAPI

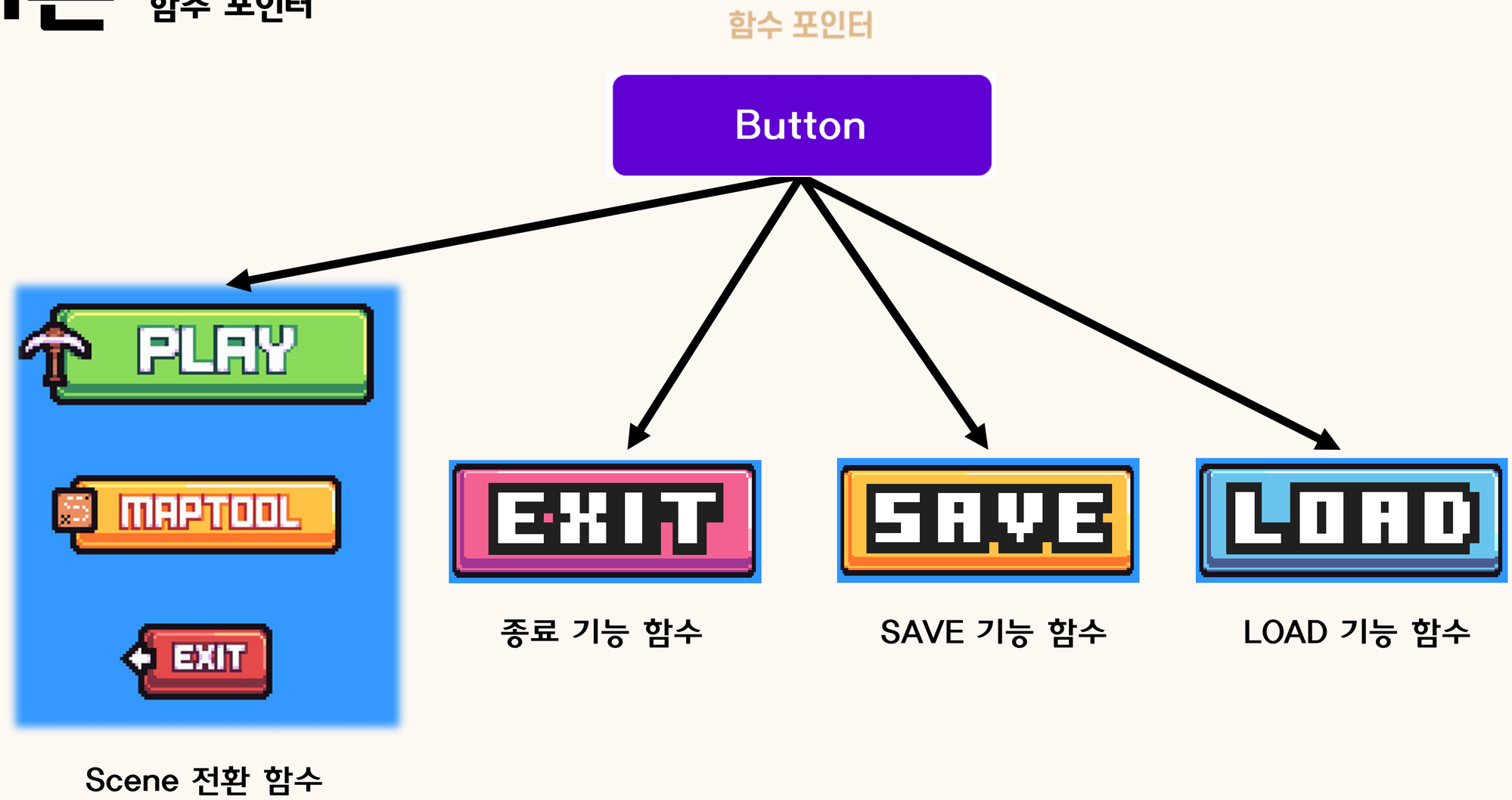
C++

GitHub

Aseprite

버튼

함수 포인터


함수 포인터
Button
PLAY
MAPTOOL
EXIT
EXIT
SAVE
LOAD
Scene 전환 함수
종료 기능 함수
SAVE 기능 함수
LOAD 기능 함수

버튼 함수 포인터

```
class Image;
class Button :public GameNode
{
private:
    BUTTON_STATE state;                 // 버튼 상태
    Image* img;                         // 버튼 이미지
    RECT rc;                            // 마우스 클릭 박스
    FPOINT pos;                         // 위치
    POINT ptDownFrame;                  // 이미지 프레임 위치
    POINT ptUpFrame;                    // 이미지 프레임 위치

    // 버튼 기능 (함수 포인터)
    void (*buttonFunc)(void*);          // 함수 포인터
    Argument_Kind kind;                 // Release를 위해 매개변수 포인터 형태를 기억하기 위한 enum
    void* arg;                          // 매개변수 포인터

public:
    HRESULT Init(const char* imageName, int posX, int posY, POINT downFramePoint, POINT upFramePoint);
    virtual HRESULT Init() override;
    virtual void Release() override;
    virtual void Update() override;
    virtual void Render(HDC hdc) override;

    void SetButtonFunc(void (*buttonFunc)(void*), Argument_Kind kind, void* arg);
    void ButtonFunc();
    void SetPos(FPOINT pos);

    FPOINT GetPos() { return pos; }

    Button();
    virtual ~Button() {}
};
```

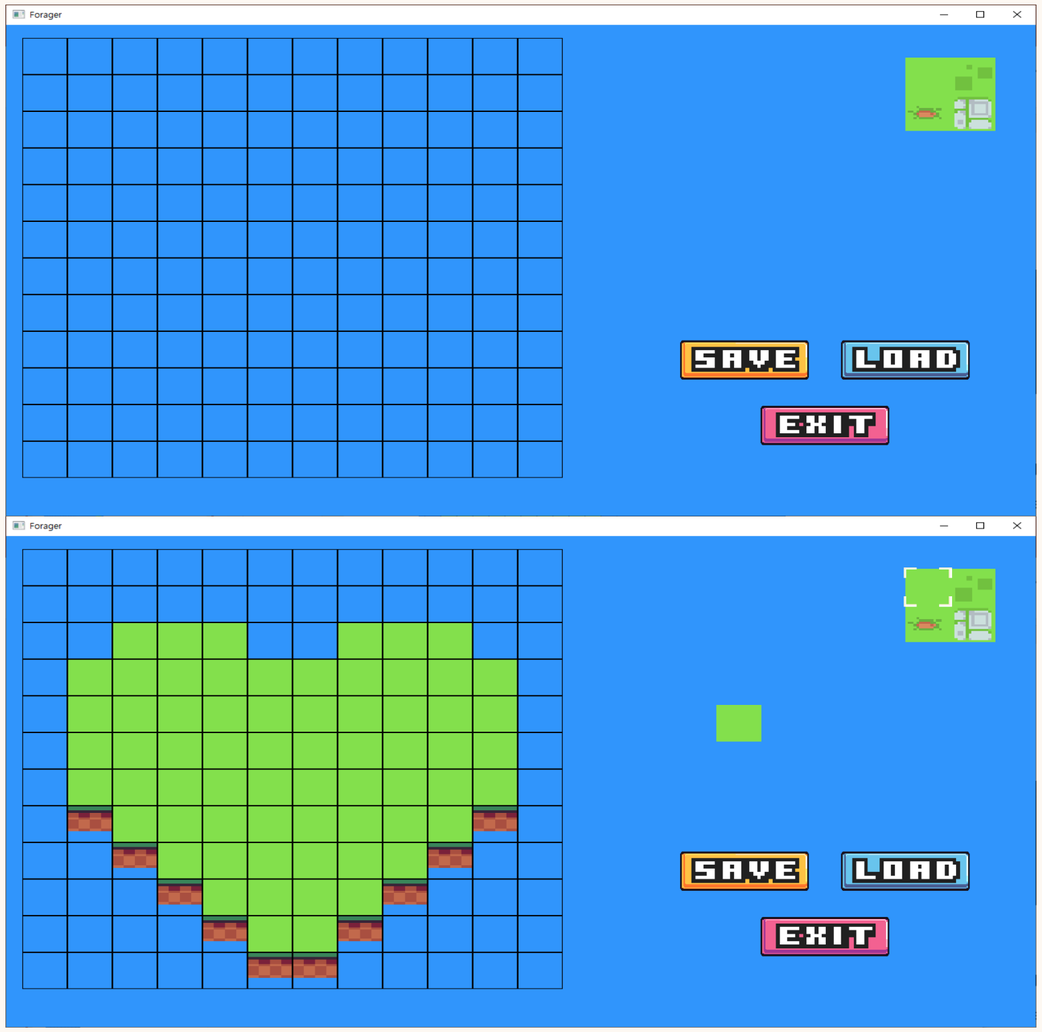
```
namespace ButtonFunction
{
    void AddValue(void* arg);
    void ChangeScene(void* arg);
    void QuitProgram(void* arg);
    void TileInfoSave(void* arg);
    void TileInfoLoad(void* arg);
}
```

버튼의 역할이 다양해서

어떤 형식이든 필요상황에 따라

넣기 위해 함수포인터를 사용하였습니다.

MAP TOOL



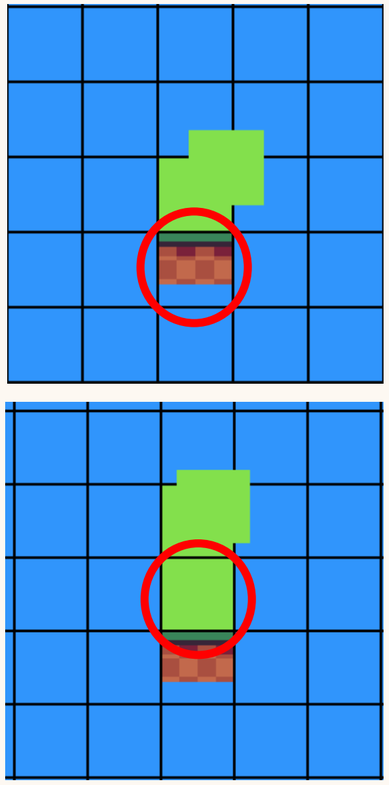

```
typedef struct tagTile
{
    TERRAIN terrain;
    Objek* object;
    POINT frame;
} TILE_INFO;
```

각 각의 타일에 구조체를 사용하여
정보를 저장합니다.

토지를 12 * 12로 규격화 하여

여러 형태의 토지를 저장하거나
불러올 수 있습니다.

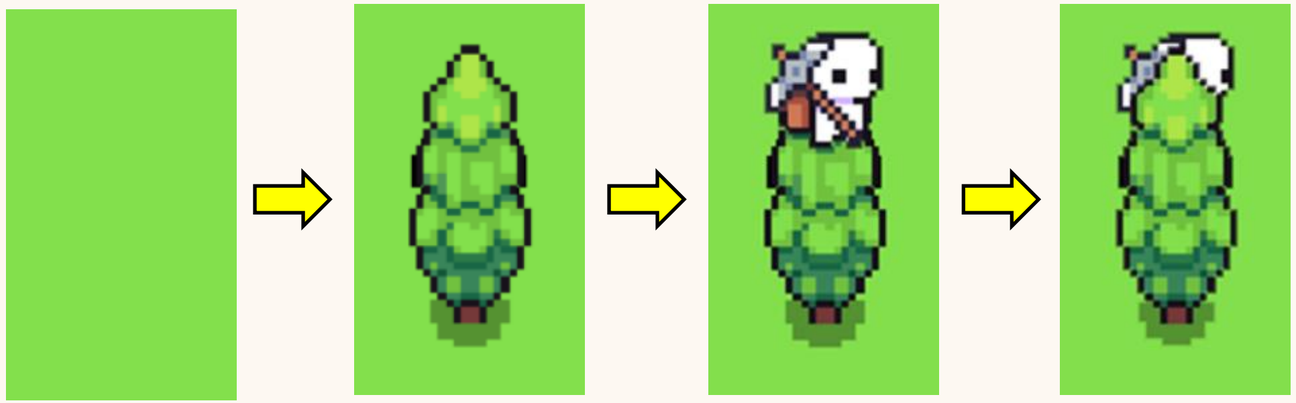
MAP TOOL



```
// 검사해서 바닥 바꿔주기
for (int y = 1; y < (TILE_Y + 1); y++)
{
    for (int x = 0; x < TILE_X; x++)
    {
        if (tileInfo[(y - 1) * TILE_X + x].terrain == LAND && tileInfo[y * TILE_X + x].terrain == WATER)
        {
            tileInfo[y * TILE_X + x].frame = sampleSubTileInfo[1].frame;
            tileInfo[y * TILE_X + x].terrain = sampleSubTileInfo[1].terrain;
        }
    }
}
```

타일의 정보가 바뀔 때 바닥 타일의 정보를 확인 후 바꿔줍니다.

Render

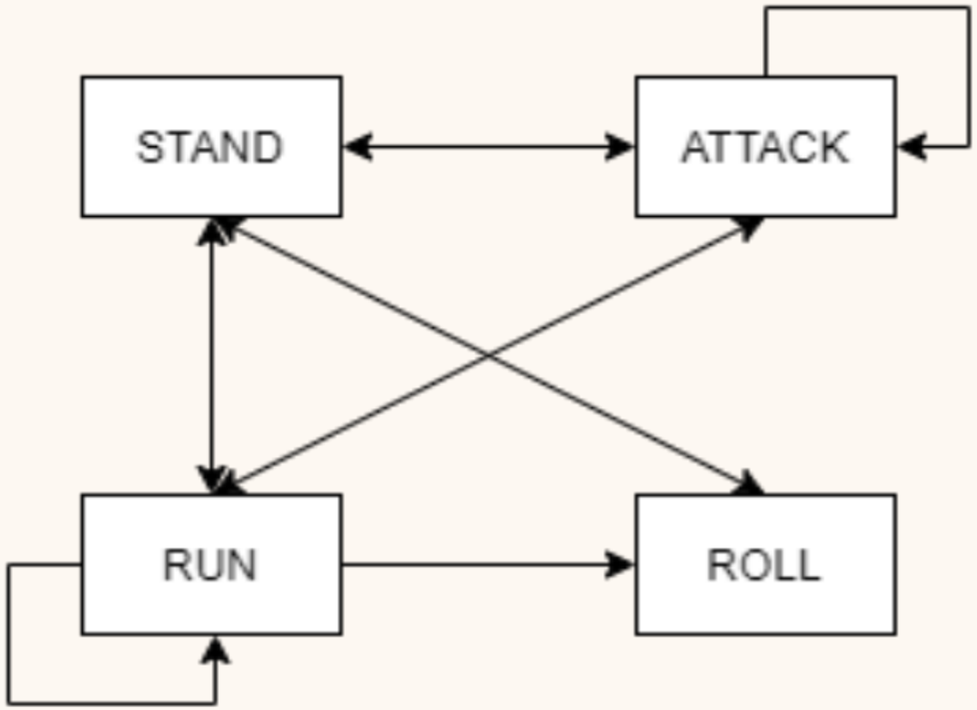


기본적인 Render 순서 타일 -> 오브젝트 -> 플레이어

플레이어가 오브젝트 뒤로 갔을 때 오브젝트보다 앞에 Render되는 현상 확인

subRender를 이용하여 플레이어의 아래에 있는 오브젝트를 검사 후 재 Render

캐릭터 상태도

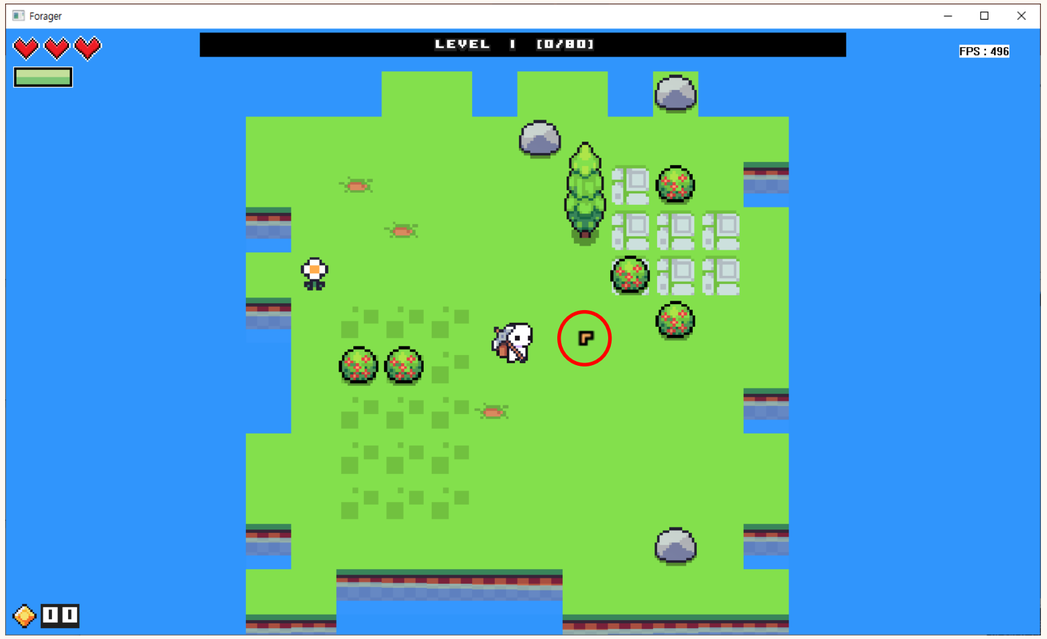


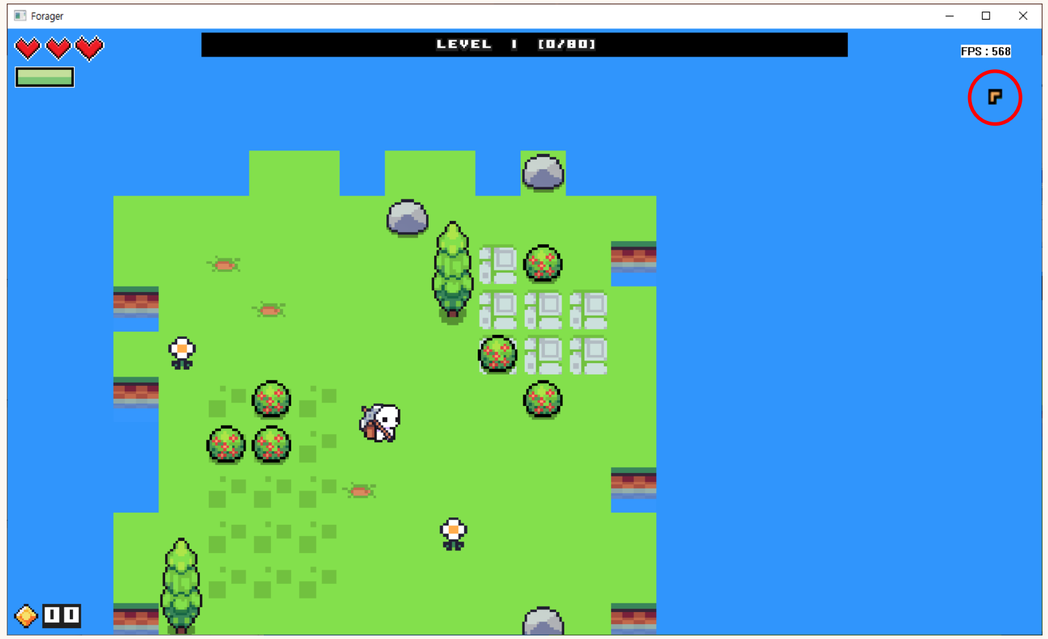

```
enum class STATE
{
    STAND,
    RUN,
    ROLL,
    ATTACK,
    END
};
```

▲ 플레이어 행동 변화 제어

CAMERA

마우스 & 플레이어 기준






```
X = target->GetPos().x - PLAYER_CENTER_X + (g_ptMouse.x - PLAYER_CENTER_X) * 0.3f;
Y = target->GetPos().y - PLAYER_CENTER_Y + (g_ptMouse.y - PLAYER_CENTER_Y) * 0.3f;
```

Player를 기준으로 Player와 마우스 위치의 비율을 7:3으로 하여 구현하였습니다.

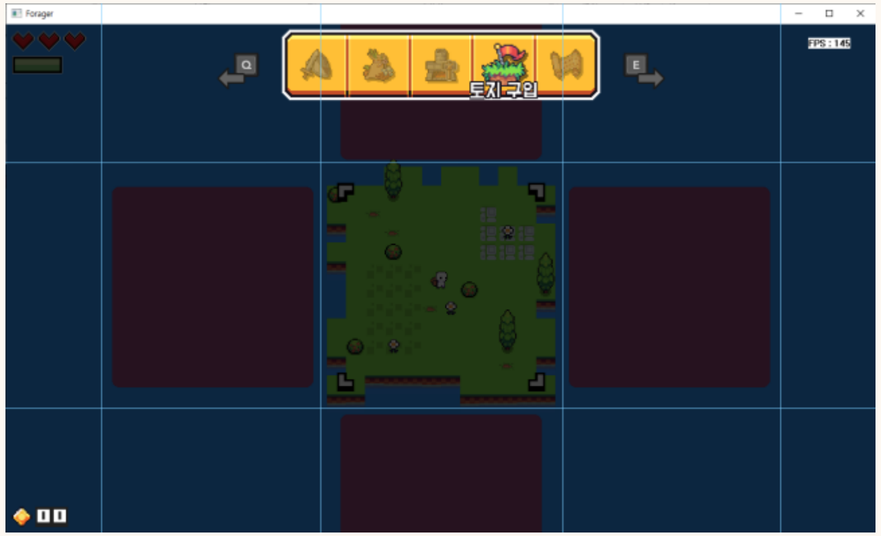
CAMERA

토지 기준

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

마우스 위치좌표를 이용해
마우스가 올라간 토지의 인덱스를 계산 후
그 토지를 화면의 기준으로 하였습니다.

* inGame화면



CAMERA

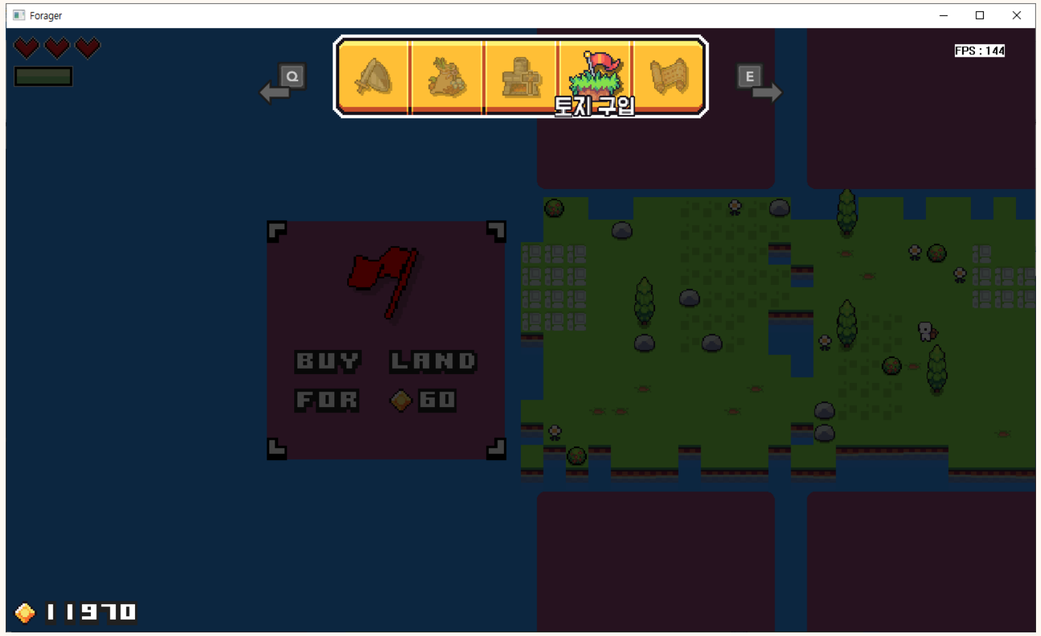
토지 기준

```
void Camera::SetCamera(float mouseX, float mouseY)
{
    int landX = mouseX / TILE_SIZE;
    int landY = mouseY / TILE_SIZE;

    if (landX < 13)
    {
        centerX = XY[0].x;
    }
    else if (landX >= 13 && landY < 25)
    {
        centerX = XY[1].x;
    }
    else if (landX >= 25 && landY < 37)
    {
        centerX = XY[2].x;
    }
    else if (landX >= 37 && landY < 49)
    {
        centerX = XY[3].x;
    }
    else if (landX >= 49)
    {
        centerX = XY[4].x;
    }
```

```
    if (landY < 13)
    {
        centerY = XY[0].y;
    }
    else if (landY >= 13 && landY < 25)
    {
        centerY = XY[5].y;
    }
    else if (landY >= 25 && landY < 37)
    {
        centerY = XY[10].y;
    }
    else if (landY >= 37 && landY < 49)
    {
        centerY = XY[15].y;
    }
    else if (landY >= 49)
    {
        centerY = XY[20].y;
    }
}
```

▲ 토지 좌표 -> 인덱스 변환

토지 구매



▲ 현재 보유한 토지를 기준으로 주변의 구매가능한 토지를 모두 표시함

다음 구매가능한 토지를 편리하게 관리하기 위하여 전체 토지를 5 * 5 형태의 인덱스로 나누었습니다.

* CAMERA - 2 참조

현재 보유중인 토지에 따라서 다음 구매가능한 토지를 나타내고 금액이 상승하는 변화를 주었습니다.

오브젝트 생성

플레이어의 현재 위치와 바다, 바닥을 제외한 토지에 무작위로 오브젝트를 생성할 수 있게 하였습니다.
각 오브젝트 별로 채집,채광,벌목에 따라 각기 다른 아이템을 드랍을 구현하였습니다.

오브젝트 생성

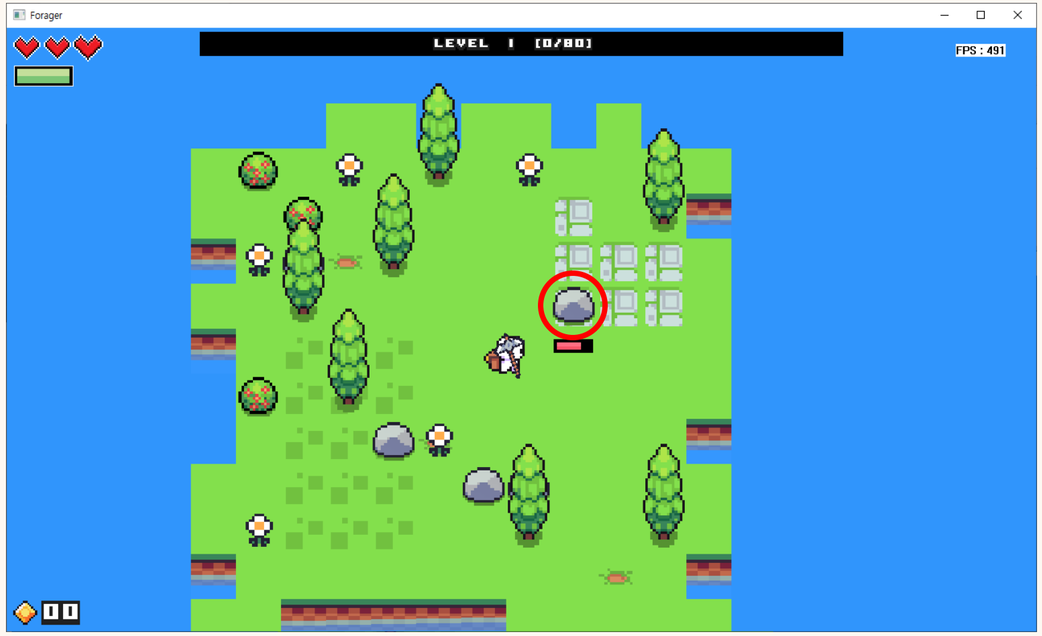
```
for (int i = 0; i < 10; i++)
{
    randNum1 = rand() % WINSIZE_TILE_MAP;
    randNum2 = rand() % WINSIZE_TILE_MAP;

    RECT tempRc;
    RECT objectRc = { (randNum2)* TILE_SIZE, (randNum1)* TILE_SIZE,
        (randNum2)* TILE_SIZE + TILE_SIZE, (randNum1)* TILE_SIZE + TILE_SIZE };

    if (tileInfo[(randNum1)* WINSIZE_TILE_MAP + (randNum2)].terrain == WATER || tileInfo[(randNum1)* WINSIZE_TILE_MAP + (randNum2)].terrain == CLIFF)
    {
        i--;
        continue;
    }
    else if (IntersectRect(&tempRc, &player->GetRc(), &objectRc))
    {
        i--;
        continue;
    }

    randOb = rand() % 4 + 1;

    if (tileInfo[(randNum1)* WINSIZE_TILE_MAP + (randNum2)].terrain == LAND)
    {
        if (tileInfo[(randNum1)* WINSIZE_TILE_MAP + (randNum2)].object != nullptr)
        {
            i--;
        }
        else
        {
            int posy = (randNum1)* TILE_SIZE;
            int posx = (randNum2)* TILE_SIZE;
            tileInfo[(randNum1)* WINSIZE_TILE_MAP + (randNum2)].object = new Objek((OBJEK)randOb, posx, posy);
            tileInfo[(randNum1)* WINSIZE_TILE_MAP + (randNum2)].object->SetPlayer(player);
            colMar->AddObjek(tileInfo[(randNum1)* WINSIZE_TILE_MAP + (randNum2)].object);
        }
    }
}
```

바다, 바닥 예외 처리

플레이어 위치 예외 처리

오브젝트

오브젝트 충돌 범위



오브젝트 충돌은 플레이어를 기준으로 하여 범위를 설정했습니다.

플레이어가 오브젝트 주변에 도달 시 충돌범위가 활성화가 됩니다.

오브젝트

오브젝트 충돌 범위

```
for (itObject = vecObject.begin(); itObject != vecObject.end(); itObject++)
{
    if ((abs)(player->GetPos().x - (g_ptMouse.x + X)) < 105 && (abs)(player->GetPos().y - (g_ptMouse.y + Y)) < 105)
    {
        if (PtInRect(&(*itObject)->GetRc(), { g_ptMouse.x + X, g_ptMouse.y + Y }))
        {
            (*itObject)->Update();
            if ((*itObject)->GetHp() <= 0)
            {
                objM->objectListInit((*itObject)->GetPoss().x, (*itObject)->GetPoss().y, (*itObject)->GetObjek());
                vecObject.erase(itObject);
                break;
            }
        }
    }
}
```

활성화

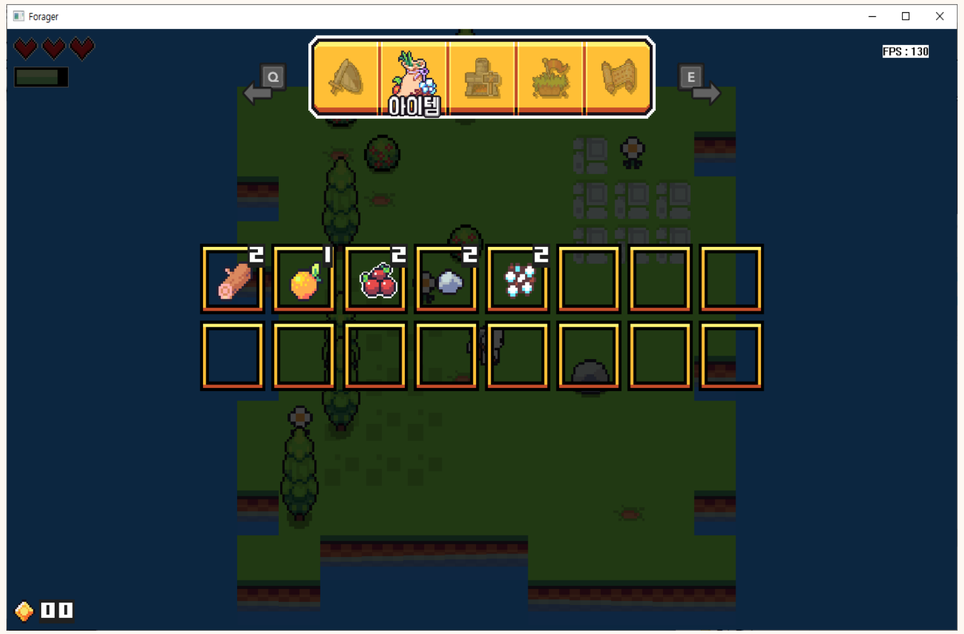
아이템 생성 및 드랍

오브젝트 해제

▲ 플레이어가 주변에 있을 때에만 활성화되는 방식

오브젝트의 체력이 0이 되었을 때 오브젝트를 해제하는 동시에 루팅 가능한 아이템을 드랍

INVENTORY




```
map< OBJECTESULT, int> inven;
```

아이템 획득 시 인벤토리의 각 슬롯에 각각의 아이템이 종류 별로 정렬하였습니다.

여러 STL중 아이템의 정보를 Key를 활용해서 찾기 위해 MAP을 사용하였습니다.

아이템 사용시 MAP의 기본기능으로 인해 자동정렬이 가능합니다.

감사합니다