計算機圖學期末專題

Group2
311551144王昇暉 311605024劉佳勳

遊戲介紹

開發環境 : Vistudio 2019

開發語言 : Open GL

遊戲內容 : 玩家需要從地圖的起點通過層層關卡移動至指定的終點

(1)路途中會有不同的地形若不慎掉落則玩家會回到起始點

(2)每段路都會貼不同的材質並且擁有不同的特性ex:冰面會滑動

(3)地圖可能會有遮擋物玩家必須旋轉視角來完成任務，操縱方向會因視角改變而不同

(4)地圖的光源會隨時間有所變化

遊戲場景

開發環境 : Vistudio 2019

開發語言 : Open GL

遊戲內容 : 玩家需要從地圖的起點通過層層關卡移動至指定的終點

(1)路途中會有不同的地形若不慎掉落則玩家會回到起始點

(2)每段路都會貼不同的材質並且擁有不同的特性ex:冰面會滑動

(3)地圖可能會有遮擋物玩家必須旋轉視角來完成任務，操縱方向會因視角改變而不同

(4)地圖的光源會隨時間有所變化

遊戲場景

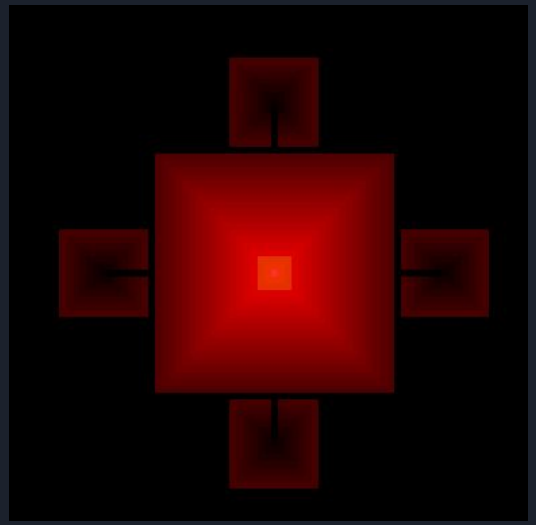
- 玩家可以自行繪製BMP來產生對應的遊完地圖

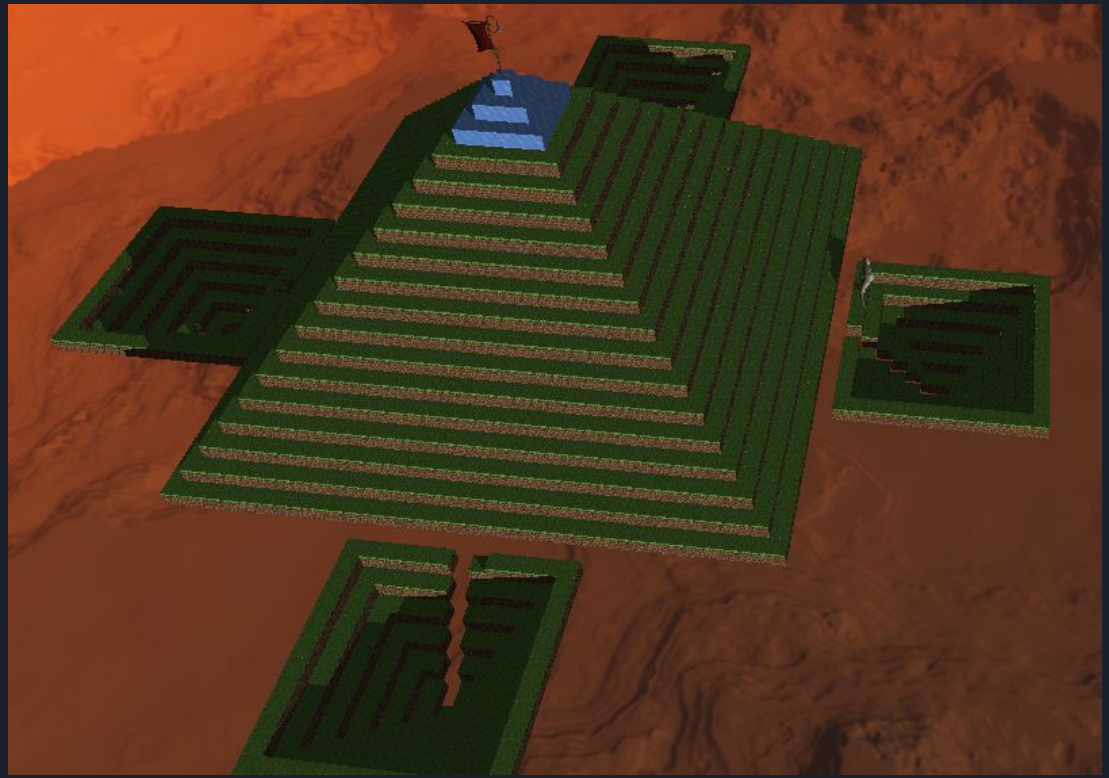
```
//載入地形 & 道具
std::string path = "../assets/map/" + std::string(mf);
const char* f = path.c_str();
struct BMPImage* mapImage = loadBMP(f);

int XOffset = mapImage->pixelWidth / 2;
int ZOffset = mapImage->pixelHeight / 2;
for (int r = 0; r < mapImage->pixelHeight; r++) {
  std::vector<float> rowHeights;
  std::vector<float> texture_idx;
  std::vector<float> props;
  for (int c = 0; c < mapImage->pixelWidth; c++) {
    if (mapImage->pixels[r * mapImage->pixelWidth + c].r != 0) {
      rowHeights.push_back(mapImage->pixels[r * mapImage->pixelWidth + c].r / BMP_HIGH_UNIT);  //高度存在紅色
      texture_idx.push_back(mapImage->pixels[r * mapImage->pixelWidth + c].g / BMP_TEX_UNIT);  //材質存在綠色
      props.push_back(mapImage->pixels[r * mapImage->pixelWidth + c].b / BMP_PROP_UNIT);       //道具存在藍色
      float x = (float)c - XOffset;
      float z = -((float)r - ZOffset);
      float y = (float)(mapImage->pixels[r * mapImage->pixelWidth + c].r) / BMP_HIGH_UNIT;
      ctx.terrainObjects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(), glm::vec3(x, y, z))));
      (*ctx.terrainObjects.rbegin())->textureIndex = texture_idx.back();
      if (props.back() == 1) {
        ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(), glm::vec3(x, y, z))));
      }
    } else {
      rowHeights.push_back(HELL);
      texture_idx.push_back(NO_TEX);
      props.push_back(NO_PROP);
    }
  }
  ctx.heightMap.push_back(rowHeights);
  ctx.textureMap.push_back(texture_idx);
  ctx.propsMap.push_back(props);
}
```

→ 讀取bmp檔案的RGB為地圖資料

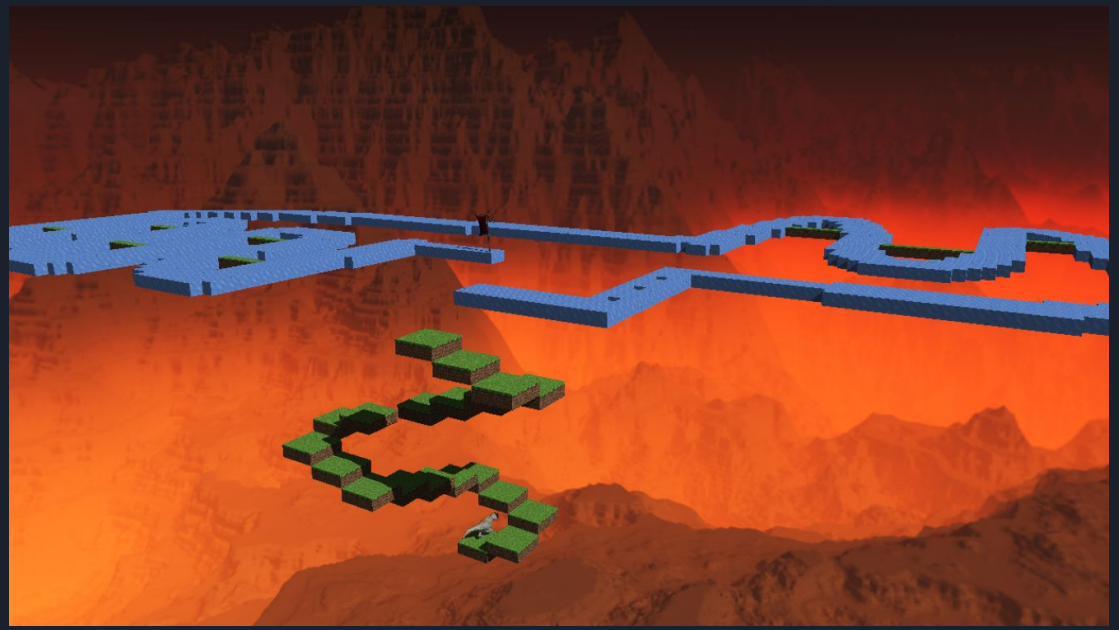
R: 地圖的height map
G: 地形的材質特性 (草地或冰地)
B: 道具種類 (終點旗)

客製化地圖

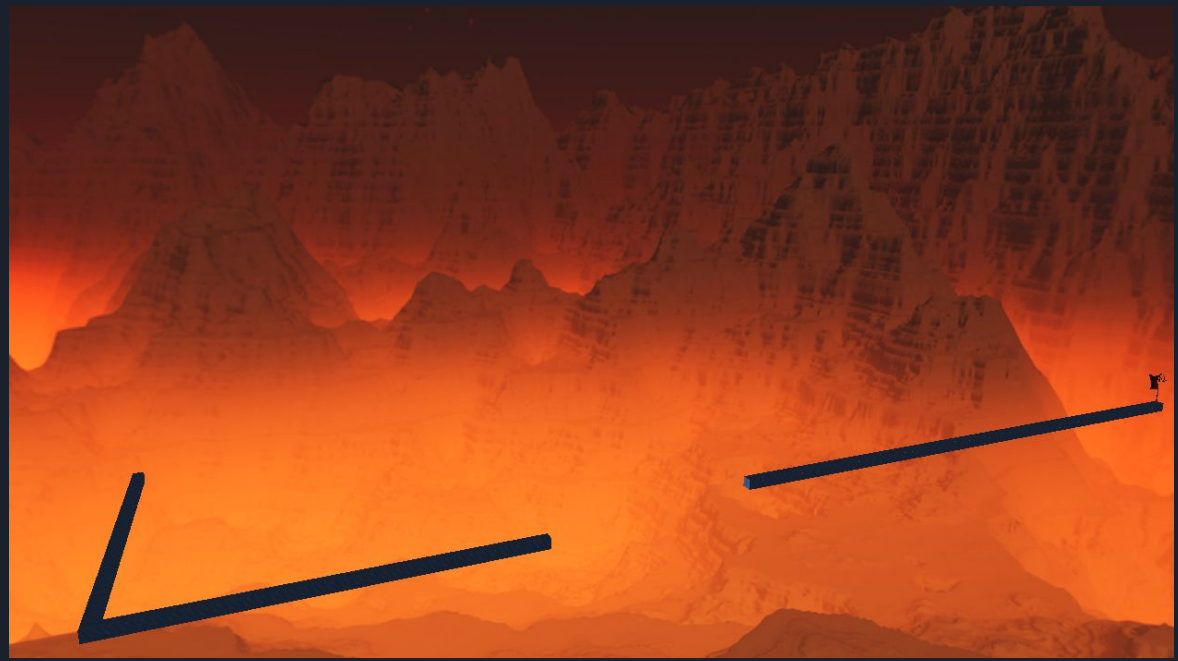




客製化地圖



客製化地圖



人物

```
class Player {
public:
  void move(bool up, bool down, bool left, bool right, float* jump_ctrl, std::vector<std::vector<float>>* heightMap, std::vector<std::vector<float>>* textureMap);
  int ismove = NOT_MOVE;
  glm::vec3 position = PLAYER_DEFAULT_POSITION;  //Player初始位置
  glm::vec3 direction = glm::vec3(0.0f, 0.0f, -1.0f); //Player方向
  glm::vec3 inertia = glm::vec3(0.0f, 0.0f, 0.0f); //Player慣性
  std::vector<Object *> objects;
  glm::mat4 rotateMatrix = glm::rotate(glm::identity<glm::mat4>(), glm::radians(180.0f), glm::vec3(0.0, 1.0, 0.0));
  glm::mat4 translateMatrix = glm::identity<glm::mat4>();
  glm::mat4 rightRotateMatrix = glm::rotate(glm::identity<glm::mat4>(), glm::radians(-PLAYER_ROTATE_SPEED), glm::vec3(0.0, 1.0, 0.0));
  glm::mat4 leftRotateMatrix = glm::rotate(glm::identity<glm::mat4>(), glm::radians(PLAYER_ROTATE_SPEED), glm::vec3(0.0, 1.0, 0.0));
  GLuint textures;
};
```

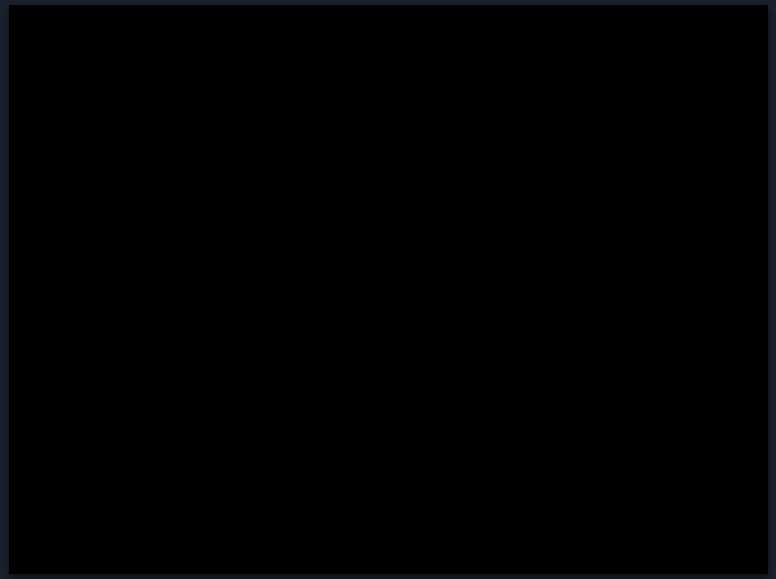
```
std::vector<std::string>player1ObjDirs(6);
player1ObjDirs[0] = "model00";
player1ObjDirs[1] = "model01";
player1ObjDirs[2] = "model02";
player1ObjDirs[3] = "model03";
player1ObjDirs[4] = "model04";
player1ObjDirs[5] = "model05";
```

→ 人物移動的object file

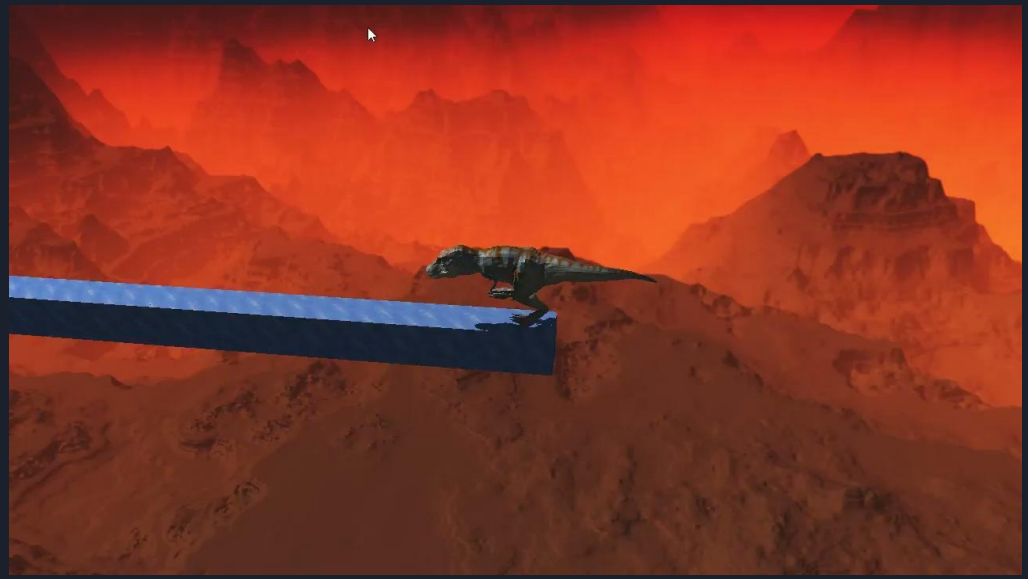
```
ctx.players.push_back(new Player());//Player 1
ctx.players[0]->textures = createTexture("../assets/models/cube/Binary_0.png");
for (int i = 0; i < 6; i++) {
  std::string path = "../assets/models/Rampaging T-Rex/" + player1ObjDirs[i] + "/model.obj";
  m = Model::fromObjectFile(path.c_str());
  m->modelMatrix = glm::scale(m->modelMatrix, glm::vec3(0.4f, 0.4f, 0.4f));
  attachGeneralObjectVAO(m);
  ctx.playerModels.push_back(m);
}
```

人物

GLB文件是以圖形語言傳輸格式（glTF）保存的3D模型，它以二進位格式存儲有關 3D模型的信息，包括節點層級、攝像機、材質、動畫和網格



轉成Obj檔讀入



人物移動 & 碰撞偵測

鍵盤X、Z軸移動

```
void Player::move(bool up, bool down, bool left, bool right, float* jump_ctrl, std::vector<std::vector<float>>* heightMap, std::vector<std::vector<float>>* textureMap) {
  glm::vec3 expected_position = position;
  int XOffset = (*heightMap)[0].size() / 2;
  int ZOffset = heightMap->size() / 2;
  int idxR, idxC;

  //鍵盤移動
  if (up) { //向前
    ismove = FORWARD;
    if (left && !right) { //左前
      rotateMatrix = leftRotateMatrix * rotateMatrix;
      direction =glm::normalize(glm::vec3(leftRotateMatrix * glm::vec4(direction, 1.0f)));
    } else if (right && !left) { //右前
      rotateMatrix = rightRotateMatrix * rotateMatrix;
      direction = glm::normalize(glm::vec3(rightRotateMatrix * glm::vec4(direction, 1.0f)));
    }
    expected_position += (direction * PLAYER_SPEED);
  } else if (down) { //向後
    ismove = BACKWARD;
    if (!left && right) {  //左後
      rotateMatrix = leftRotateMatrix * rotateMatrix;
      direction = glm::normalize(glm::vec3(leftRotateMatrix * glm::vec4(direction, 1.0f)));
    } else if (!right && left) {  //右後
      rotateMatrix = rightRotateMatrix * rotateMatrix;
      direction = glm::normalize(glm::vec3(rightRotateMatrix * glm::vec4(direction, 1.0f)));
    }
    expected_position -= (direction * PLAYER_SPEED);
  } else if (left && !right) {  //左轉
    ismove = TURN_LEFT;
    rotateMatrix = leftRotateMatrix * rotateMatrix;
    direction = glm::normalize(glm::vec3(leftRotateMatrix * glm::vec4(direction, 1.0f)));
  } else if (right && !left) {  //右轉
    ismove = TURN_RIGHT;
    rotateMatrix = rightRotateMatrix * rotateMatrix;
    direction = glm::normalize(glm::vec3(rightRotateMatrix * glm::vec4(direction, 1.0f)));
  } else {
    ismove = NOT_MOVE;
  }
```

人物移動 & 碰撞偵測

X、Z軸碰撞偵測以及移動效果

```
//慣性加速
if ((ismove == NOT_MOVE) && (inertia != glm::vec3(0.0f, 0.0f, 0.0f))) ismove = SLIDE;
expected_position += inertia;

//前後碰撞偵測
idxC = round((expected_position.x + XOffset));
idxR = round(-expected_position.z + ZOffset);
if (((*heightMap)[idxR][idxC] - expected_position.y) > 0 && ((*heightMap)[idxR][idxC] - expected_position.y) <= 2) { // 0 < 高度差 <= 2
  //遭遇阻擋
  idxC = round((position.x + XOffset));
  idxR = round(-position.z + ZOffset);
} else {
  //移動成功
  position = expected_position;

  if ((*heightMap)[idxR][idxC] == position.y) { //如果站在地面上
    if ((*textureMap)[idxR][idxC] == ICE) { //冰面
      //移動產生慣性
      if (ismove == FORWARD) {
        inertia += (direction * INERTIA_ACC);
      } else if (ismove == BACKWARD) {
        inertia -= (direction * INERTIA_ACC);
      }
      //摩擦力減速
      if (glm::length(inertia) > 0.0f) {
        inertia -= (glm::normalize(inertia) * FRICTION);
      }
    } else {
      inertia = glm::vec3(0.0f, 0.0f, 0.0f);
    }
  }
}
```

人物移動 & 碰撞偵測

Y軸碰撞偵測以及移動效果

```
if (jump_ctrl[VELOCITY_Y] != 0) ismove = JUMP;  //有Y軸初速

if (position.y > (*heightMap)[idxR][idxC]) { //比地板高才會變地板
    jump_ctrl[FLOOR] = (*heightMap)[idxR][idxC];
} else if (position.y < (*heightMap)[idxR][idxC]) { //比地板低落下
    jump_ctrl[FLOOR] = HELL;
}

position.y += ((jump_ctrl[VELOCITY_Y] * TIME_CUT) - 0.5 * G * TIME_CUT * TIME_CUT);
jump_ctrl[VELOCITY_Y] = jump_ctrl[VELOCITY_Y] - G * TIME_CUT;

if (jump_ctrl[FLOOR] == HELL) {
    //撞到上層地板
    if ((((*heightMap)[idxR][idxC] - MAP_THICKNESS) - (position.y + PLAYER_TALL)) > 0 &&
        (((*heightMap)[idxR][idxC] - MAP_THICKNESS) - (position.y + PLAYER_TALL)) < 0.05) {
        jump_ctrl[VELOCITY_Y] = V_REFLECT;
    }
}

if (position.y <= jump_ctrl[FLOOR]) { //落到地板
    jump_ctrl[CTRL] = DISABLE_JUMP;
    jump_ctrl[VELOCITY_Y] = 0;
    position.y = jump_ctrl[FLOOR];
}

if (position.y == HELL) { //墜入深淵
    position = PLAYER_DEFAULT_POSITION;  // Player初始位置
    inertia = glm::vec3(0.0f, 0.0f, 0.0f);
}

translateMatrix = glm::translate(glm::identity<glm::mat4>(), position);
```

人物移動 & 碰撞偵測

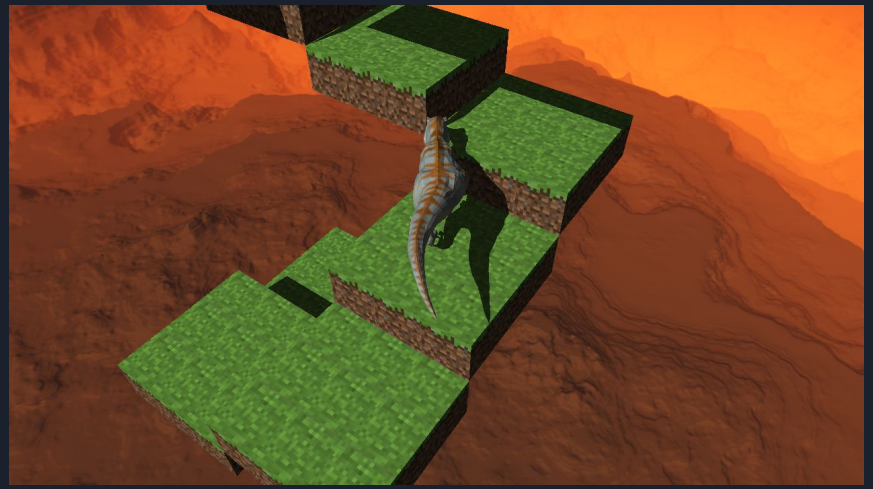
X、Z軸地形偵測

Y軸地形偵測

視角

當玩家按下案鍵L時可以切換遊玩視角

鎖定遊玩視角視角時玩家會相機會對準人物並尾隨，按下W可以拉近，按下A可以拉遠

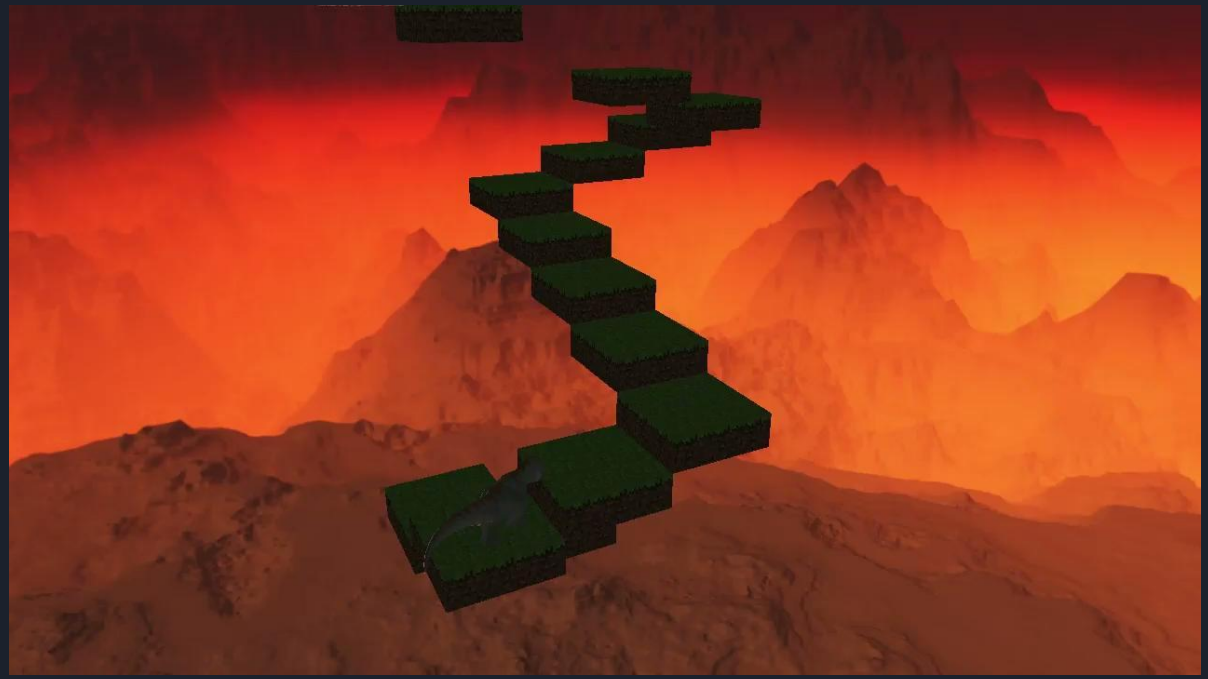


上帝視角可以使用滑鼠及 WASD操控相機

光影

```
//光源移動
if (ctx.lightDegree < 180) {
  ctx.lightDegree += LIGHT_SPEED;
} else {
  ctx.lightDegree += (LIGHT_SPEED * DAY_NIGHT_RATIO);
  if (ctx.lightDegree >= 360) ctx.lightDegree = 0;
}
ctx.lightDirection = glm::vec3(-0.3, -0.3 * sinf(glm::radians(ctx.lightDegree)), -0.3 * cosf(glm::radians(ctx.lightDegree)));
```

實作平行光源移動，模擬日出日落的光影的變化

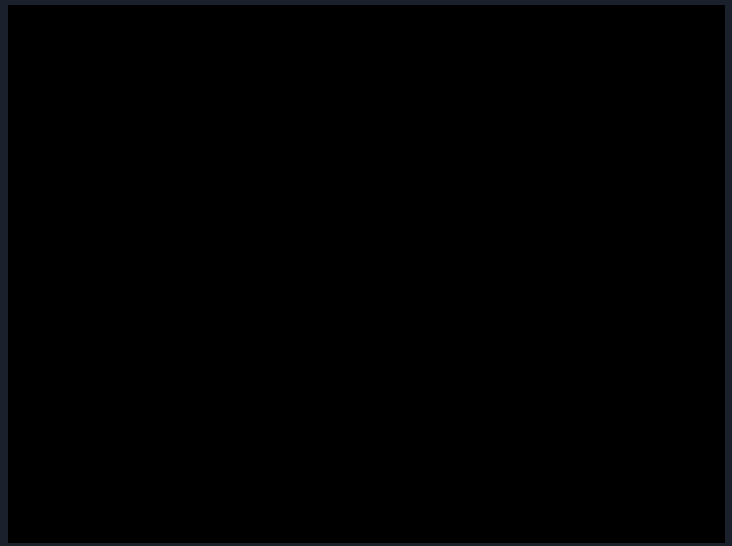


場景效果

```
//濾鏡變換
if (ctx.players[0]->position.y < BLACK_SCENE_THRESHOLD) {
  ctx.eanbleBlackscene = (ctx.players[0]->position.y - HELL) / (BLACK_SCENE_THRESHOLD - HELL);
}
if (ctx.players[0]->position == PLAYER_DEFAULT_POSITION) ctx.eanbleBlackscene = 0;

if (glm::length(ctx.players[0]->inertia) < SPEED_THRESHOLD1) {
  ctx.eanbleSpeed = SLOW_SPEED;
} else if (glm::length(ctx.players[0]->inertia) < SPEED_THRESHOLD2) {
  ctx.eanbleSpeed = MID_SPEED;
} else {
  ctx.eanbleSpeed = FAST_SPEED;
}
```

墜落場景

高速場景



材質特性

在冰面會有摩擦力小，物體的慣性 產生打滑的效果

參數控制

```
/*Player control*/
#define PLAYER_DEFAULT_POSITION glm::vec3(0.0f, 2.0f, 0.0f)
#define PLAYER_TALL (1.7f)

/*Player moving control*/
#define G (9.8) //重力加速度
#define TIME_CUT (0.01f) //跌落時間控制
#define PLAYER_SPEED (0.025f) //移動速度
#define INERTIA_ACC (PLAYER_SPEED * (0.01f)) //慣性加速度
#define FRICTION (0.0001f)  //摩擦力係數
#define PLAYER_ROTATE_SPEED (0.5f) //轉動速度

//speed
#define SLOW_SPEED 0  //慢速
#define MID_SPEED 1  //中等速度
#define FAST_SPEED 2 //快速

#define UP 0 //案鍵上
#define DOWN 1 //案鍵下
#define LEFT 2 //案鍵左
#define RIGHT 3 //案鍵右

//ismove
#define NOT_MOVE 0 //沒移動
#define FORWARD 1 //人物向前
#define BACKWARD 2 //人物向後
#define TURN_LEFT 3 //人物左轉
#define TURN_RIGHT 4  //人物左轉
#define JUMP 5 //人物跳
#define SLIDE 6 //滑行
#define LOCK 10 //鎖定視角

//jump_ctrl
#define CTRL 0  //控制是否是跌落狀態
#define DISABLE_JUMP 0
#define FIRST_JUMP 1
#define SECOND_JUMP 2
#define VELOCITY_Y 1 //人物Y軸移動速度
#define V0 6  //初速(用於跳躍高度控制)
#define V_REFLECT (-2) //狀頂反彈速度
```

```
#define FLOOR 2 //地板

#define HELL (-100.0f) //跌落高度

/*Camera control*/
#define NOT_LOCK_VIEW 0
#define LOCK_VIEW 1
#define DEFAULT_DISTANCE (7.5f)
#define CAMERA_DEFAULT_POSITION glm::vec3(0, 4, 8)

/*Map control*/
#define MAP_0 0 //第0張地圖
#define MAP_1 2
#define MAP_2 4
#define BMP_HIGH_UNIT 10 //每一高度色差
#define MAP_THICKNESS 1
#define BMP_TEX_UNIT 50 //不同材質色差
#define GRASS 0
#define ICE 1
#define MUD 2
#define NO_TEX 10
#define BMP_PROP_UNIT 50 //不同道具色差
#define NO_PROP 0
#define FLAG 1

/*Light control*/
#define LIGHT_SPEED (0.05f)
#define DAY_NIGHT_RATIO 2

/*Filter control*/
#define BLACK_SCENE_THRESHOLD (HELL + 90.0f)  //黑畫面高度
```

Demo