PPGSO PROJEKT

Správa k projektu

Peter Bartoš - **xbartosp2@stuba.sk**
Jakub Grúber - **xgruber@stuba.sk**

Čas cvičení: Utorok 16:00 – 18:00

Cvičiaci: Ing. Peter Kapec Phd.

Dátové štruktúry

Dátové štruktúry objektov v našom projekte sú nasledovné:

```
cat::cat(glm::vec3 pos, glm::vec3 rot, glm::vec3 scl) {
    position = pos;
    rotation = rot;
    scale = scl;
    if (!shader) shader = std::make_unique<ppgso::Shader>( vertex_shader_code: phong_vert_glsl,  fragment_shader_code: phong_frag_glsl);
    if (!shadowShader) shadowShader = std::make_unique<ppgso::Shader>( vertex_shader_code: shadow_map_vert_glsl,  fragment_shader_code: shadow_map_frag_glsl);
    if (!texture) texture = std::make_unique<ppgso::Texture>( image: ppgso::image::loadBMP( bmp: "cat.bmp"));
    if (!mesh) mesh = std::make_unique<ppgso::Mesh>( obj: "cat.obj");
};
```

Každý objekt má takýto konštruktor. Má premennú na pozíciu, rotáciu a škálovanie, ktoré sú všetky 3D vektory, teda majú súradnice x, y a z. Ďalej sú ukladané shadre na svetlo a tiene. Potom nasleduje textúra a mesh, teda „telo“ objektu.

Objekty majú taktiež dátovú štruktúru na ukladane pointerov na svoje deti a taktiež majú pointre na svojich rodičov. Pointre na rodičov sú dva, pretože ak rodič objektu je samotná scéna, ktorá nie je triedy Object, tak pointer musí ukazovať na premennú typu Scene. Týmto (zoznamom detí a pointrami na rodičov) je zabezpečený graf scény, kde v potomkovi vieme získať údaje jeho predkov.

```
class cat final : public Object {
private:
    static std::unique_ptr<ppgso::Mesh> mesh;
    static std::unique_ptr<ppgso::Shader> shader;
    static std::unique_ptr<ppgso::Shader> shadowShader;
    static std::unique_ptr<ppgso::Texture> texture;
    Object* parent = nullptr;
    Scene * sceneParent = nullptr;
    std::list<std::unique_ptr<Object>> children;
```

Ďalšími zaujímavými dátovými štruktúrami sú struct keyFrame a cameraAnimation:

```
struct keyFrame {
    glm::vec3 position;
    glm::vec3 lookAt;
    float time;
};

struct cameraAnimation {
    std::vector<keyFrame> keyFrames;

    cameraAnimation(const std::vector<keyFrame>& frames)
            : keyFrames(frames) {}
};
```

Struct keyFrame reprezentuje pozíciu, miesto pohľadu kamery a čas, kedy sa má kamera na danom mieste nachádzať. CameraAnimation združuje všetky kľúčové snímky kamery, ktoré sú používané pri prelete kamery scénou.

Tak isto animovaný objekt ako napríklad delfín má dátovú štruktúru pre ukladanie kľúčových snímkov animácie:

```
struct dolphinKeyFrame {
    glm::vec3 position;
    glm::vec3 rotation;
    float time;
};

std::vector<dolphinKeyFrame> keyframes;
```

Podobne ako pri kamere, v samotnom sníku sa ukladá pozícia, rotácia a čas daného kľúčovému snímku. Taktiež je definovaný vektor, ktorý v sebe udržuje všetky snímky animácie delfína. Animácia tak vzniká interpolovaním medzi kľúčovými snímkami.

Zaujímavé algoritmy

Jedným z algoritmov v našom projekte je napríklad algoritmus na generovanie Bézierovej krivky pre animáciu delfína. Táto animácia je zadefinovaná 4 kontrolnými bodmi.
V nasledujúcej funkcií sa generujú body medzi zadanými 4 kontrolnými bodmi a vygeneruje sa 30 bodov, ktoré tvoria finálnu krivku, po ktorej delfín „skáče“.

```
void dolphin::generateBezierKeyframes(const glm::vec3& start_position, const glm::vec3& start_rotation, float start_time,
                          const glm::vec3& control1_position, const glm::vec3& control1_rotation, float control1_time,
                          const glm::vec3& control2_position, const glm::vec3& control2_rotation, float control2_time,
                          const glm::vec3& end_position, const glm::vec3& end_rotation, float end_time, int segments) {
    for (int i = 0; i <= segments; ++i) {
        float t = static_cast<float>(i) / static_cast<float>(segments);
        float u = 1.0f - t;

        glm::vec3 position =
                u * u * u * start_position + 3.0f * u * u * t * control1_position
                + 3.0f * u * t * t * control2_position + t * t * t * end_position;

        glm::vec3 rotation =
                u * u * u * start_rotation + 3.0f * u * u * t * control1_rotation
                + 3.0f * u * t * t * control2_rotation + t * t * t * end_rotation;

        float time =
                u * u * u * start_time + 3.0f * u * u * t * control1_time
                + 3.0f * u * t * t * control2_time + t * t * t * end_time;

        addKeyFrame( keyframe: {position, rotation, time});
    }
};
```

Body sa tvoria na základe tohto explicitného vzorca pre Bézierovu krivku:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3,\ 0 \leq t \leq 1.$$

Následne sa delfínovi vo funkcií update generuje zostavuje finálna matica, ktorá hovorí, kde sa delfín vykreslí:

```
bool dolphin::update(Scene &scene, float dt) {

    dolphinKeyFrame startFrame, endFrame;

    for (size_t i = 0; i < keyframes.size() - 1; ++i) {
        if (currentTime >= keyframes[i].time && currentTime < keyframes[i + 1].time) {
            startFrame = keyframes[i];
            endFrame = keyframes[i + 1];
            break;
        }
    }

    float t = glm::clamp( x: (currentTime - startFrame.time) / (endFrame.time - startFrame.time),  minVal: 0.0f,  maxVal: 1.0f);
    position = glm::mix( x: startFrame.position,  y: endFrame.position,  a: t);
    rotation = glm::mix( x: startFrame.rotation,  y: endFrame.rotation,  a: t);

    currentTime += dt;

    generateModelMatrix();

    return true;
}
```

Najprv sa podľa času nájdu dva snímky, medzi ktorými bude delfín intepolovať. Potom sa vypočíta parameter t nasledovne:

(currentTime - startFrame.time) -> Táto časť vypočíta čas, ktorý uplynul od začiatku aktuálneho intervalu kľúčových snímok. currentTime je aktuálny čas a startFrame.time je čas začiatku kľúčovej snímky.

(endFrame.time - startFrame.time) -> Táto časť vypočíta celkové trvanie aktuálneho intervalu kľúčových snímok.

Hodnota je nakoniec clampnutá na interval od 0 do 1, aby faktor nevyšiel z valídneho rozsahu.

Pozícia a rotácia sa potom vypočítajú funkciou glm::mix, ktorá robí lineárnu interpoláciu medzi dvoma kľúčovými snímkami za pomoci parametra t.

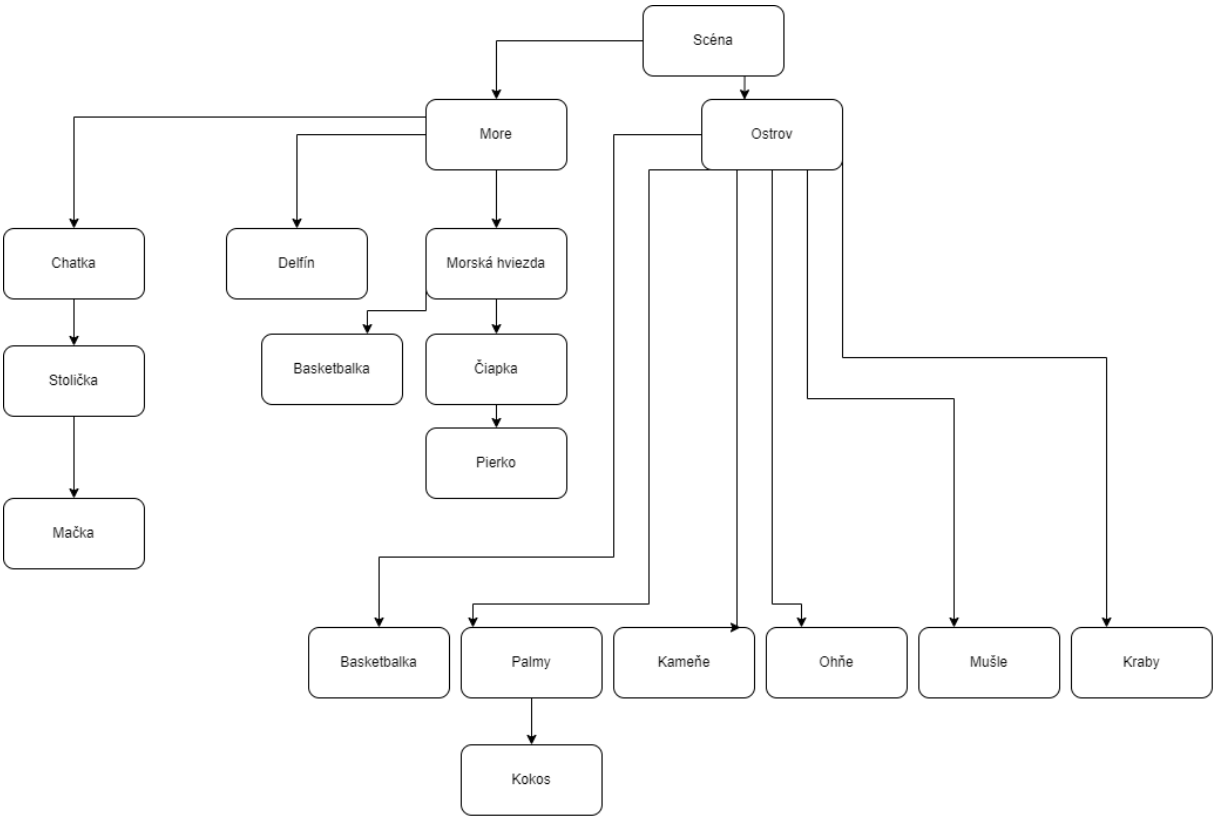
Ďalším zaujímavým algoritmom je napríklad algoritmus na generovanie paliem na ostrove:

```
void island::generatePalms(glm::vec3 pos, glm::vec3 scl) {
    auto palm_points = {/*center ring*/44, 45, 54, /*1st ring*/34, 35, 43,
46, 53, 56, 64, 65, /*2nd ring*/24, 37, 62, 74};
    int j = 0;
    for (auto point : palm_points) {
        auto test_palm = std::make_unique<palm>(vertices[point] * scl +
pos,rotation,glm::vec3{0.05, random_number(0.04,0.05),0.05}, false);
        int baby_palms = 2;
        if (j > 2) baby_palms = 4;
        for (int i = 0; i < baby_palms; ++i) {
            glm::vec3 offset = {7 * cos(i * M_PI / 2.0), random_number(-1,
-6), 7 * sin(i * M_PI / 2.0)};
            glm::vec3 r_scale = glm::vec3{random_number(0.04, 0.05),
random_number(0.04, 0.05),
                                          random_number(0.04, 0.05)};
            test_palm = std::make_unique<palm>(vertices[point] * scl + pos
+ offset, glm::vec3{0,0,random_number(-ppgso::PI, ppgso::PI)}, r_scale,
false);
            addChild(std::move(test_palm));
        }
        j++;
    }
    glm::vec3 main_palm_pos =
calculateSquareCenter(vertices[44],vertices[45],vertices[54],vertices[55]);
    auto test_palm = std::make_unique<palm>(main_palm_pos * scl +
pos,rotation,glm::vec3{0.06, 0.07, 0.06}, true);
    addChild(std::move(test_palm));
}
```

Vstupom algoritmu je pozícia a rotácia ostrova. Následne sme vybrali body na ostrove, kde budú palmy. Potom sa v cykle prechádza týmito bodmi. Na danom bode sa najprv vytvorí jedna palma s náhodnou veľkosťou v zadanom rozmedzí. Potom sa okolo nej idú generovať ďalšie palmy, ktoré budú mať náhodný offset od prvej palmy na bode ostrova, náhodné natočenie a náhodnú veľkosť. Na záver sa vytvorí veľká palma, ktrorá je v strede ostrova, resp. na pozícií medzi 4 bodmi na vrchole ostrova, z ktorej ako jedinej bude padať kokos.

Opis scény pomocou grafu scény

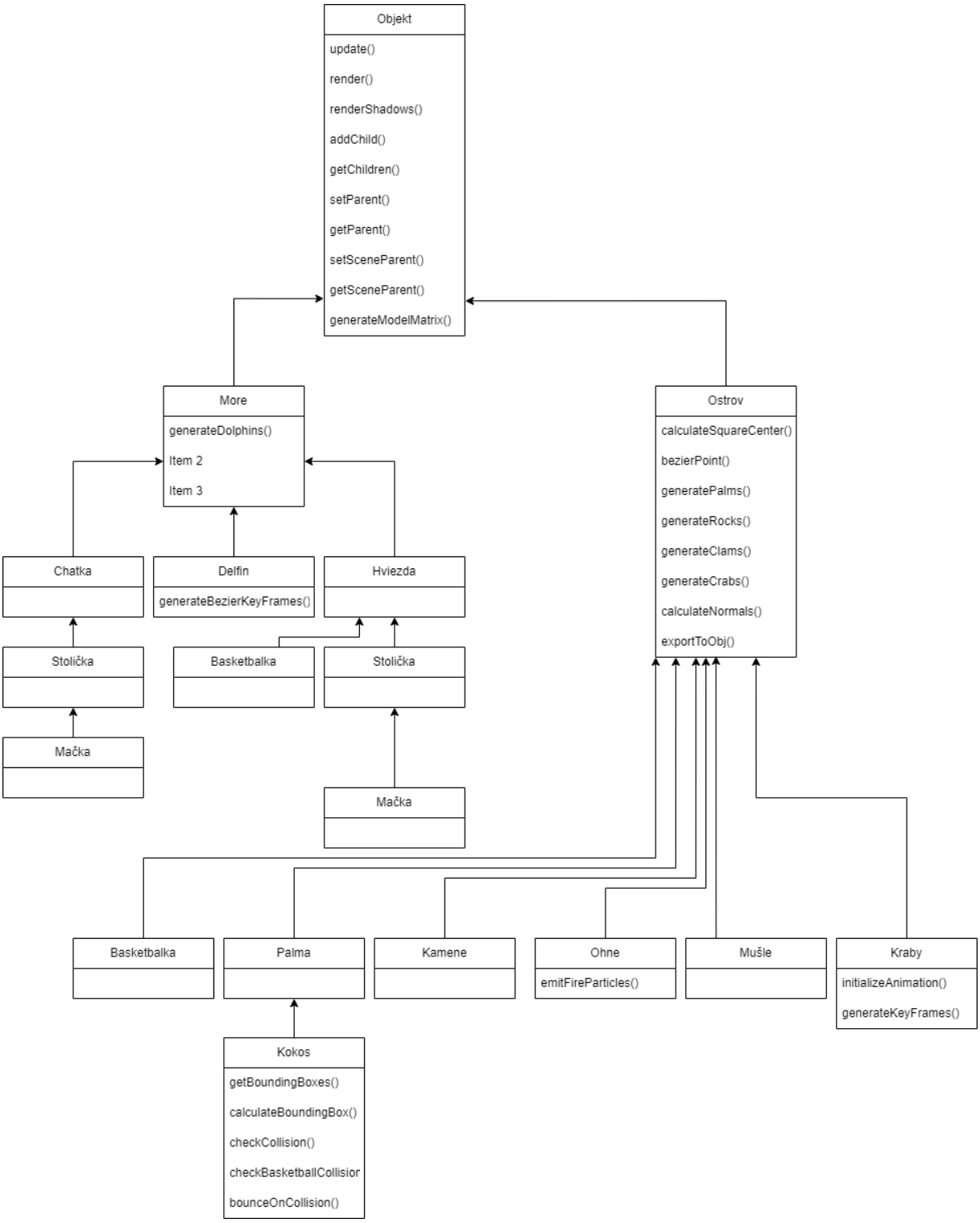
V našom projekte máme jednu scénu, na ktorej sa nachádzajú objekty. Prestavujú ju more a ostrov, na ktorom sa nachádzajú objekty. V rámci tejto scény máme aj vnútornú oblasť, ktorú predstavuje interiér chatky. Graf scény vyzerá nasledovne:



V scéne máme more, v ktorom skáče delfín. Nad morom lieta svietiaca morská hviezda, ktorá má klobúk s pierko a lieta okolo nej basketbalová lopta. Potom máme Chatku, ktorá je tiež potomkom mora, v ktorej je húpajúca sa stolička, na ktorej sedí mačka. Nakoniec máme ostrov, na ktorom sú kameňe, mušle, palmy, kraby, ohňe a basketbalová lopta. Z jednej palmy v strede ostrova padá kokos.

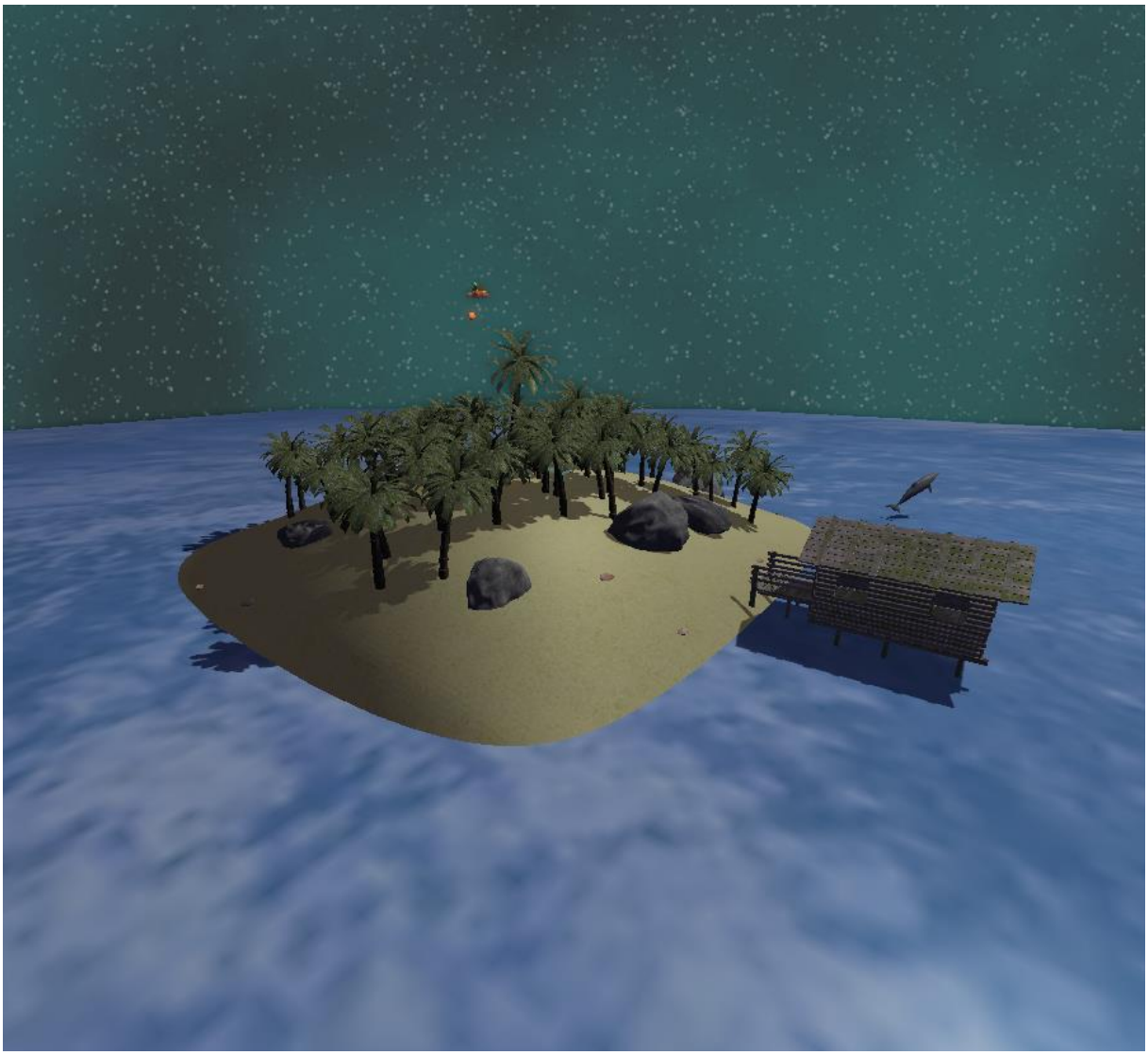
Opis objektov v scéne diagramom tried

V nasledujúcom diagrame sú opísané objekty. V triede Objekt sú spoločné funkcie každého objektu, v ostatných triedach sú spomenuté len tie, ktoré sú unikátne.

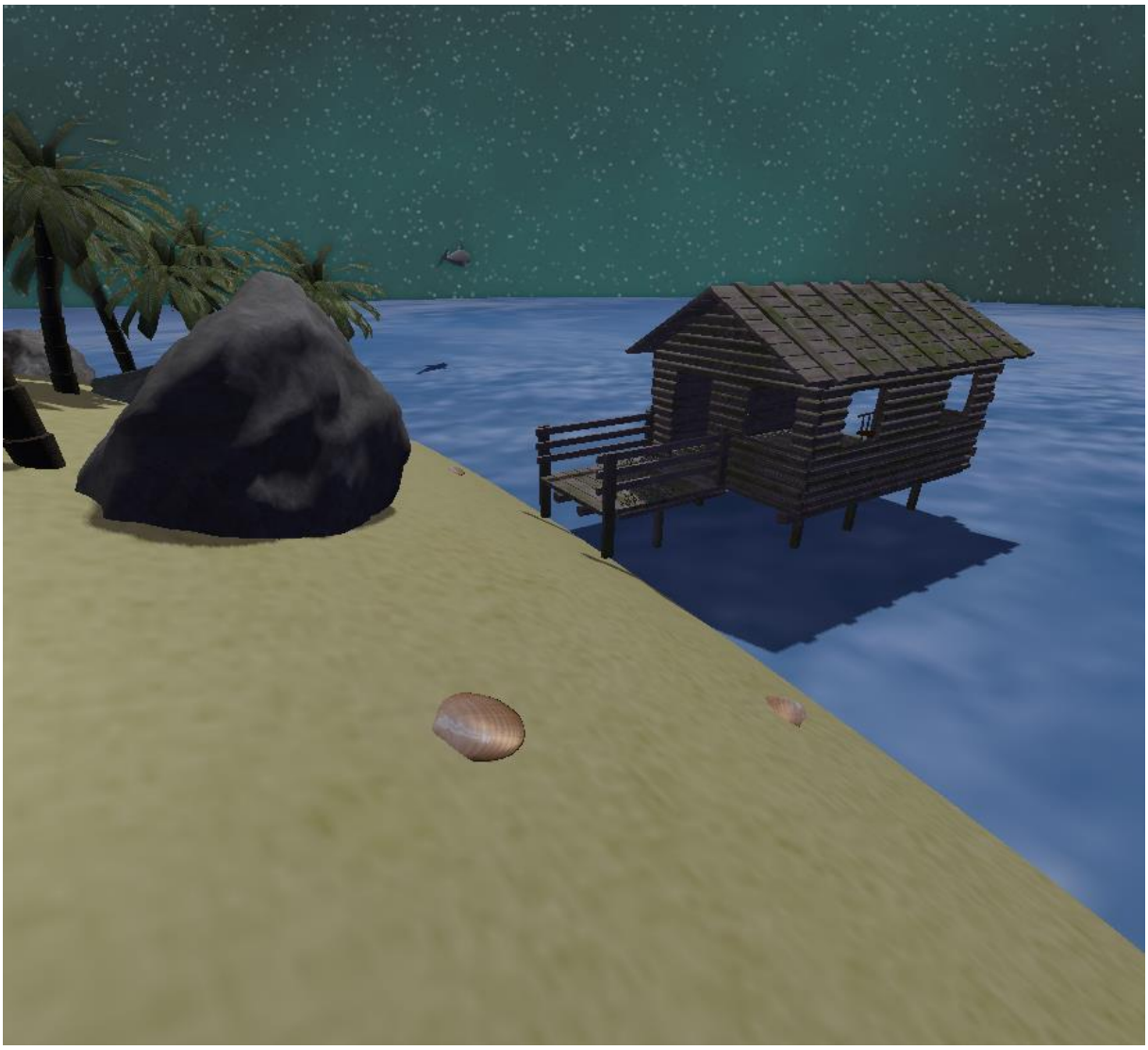


Screenshoty z projektu

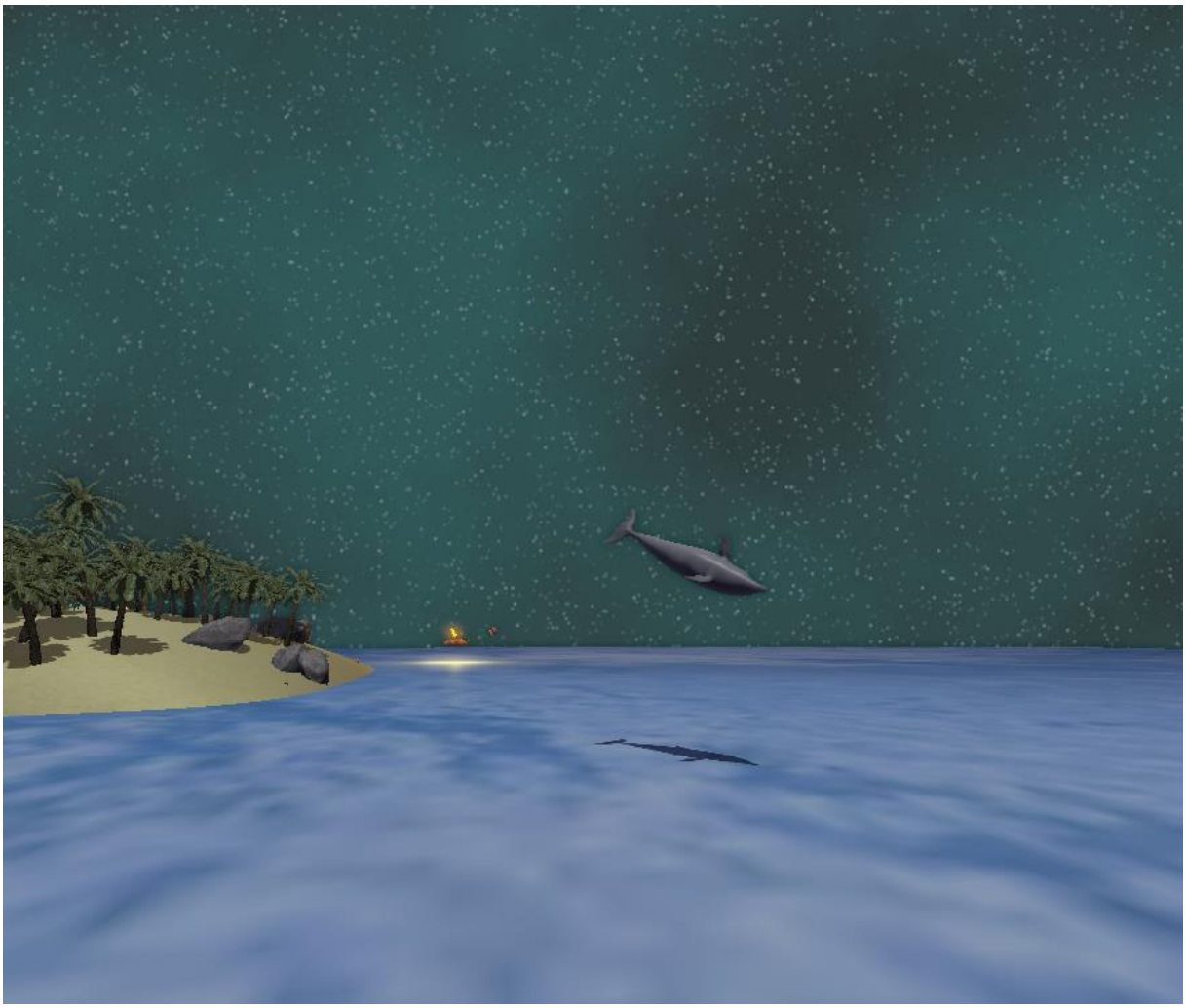
Pohľad na celý ostrov



Pohľad na chatku

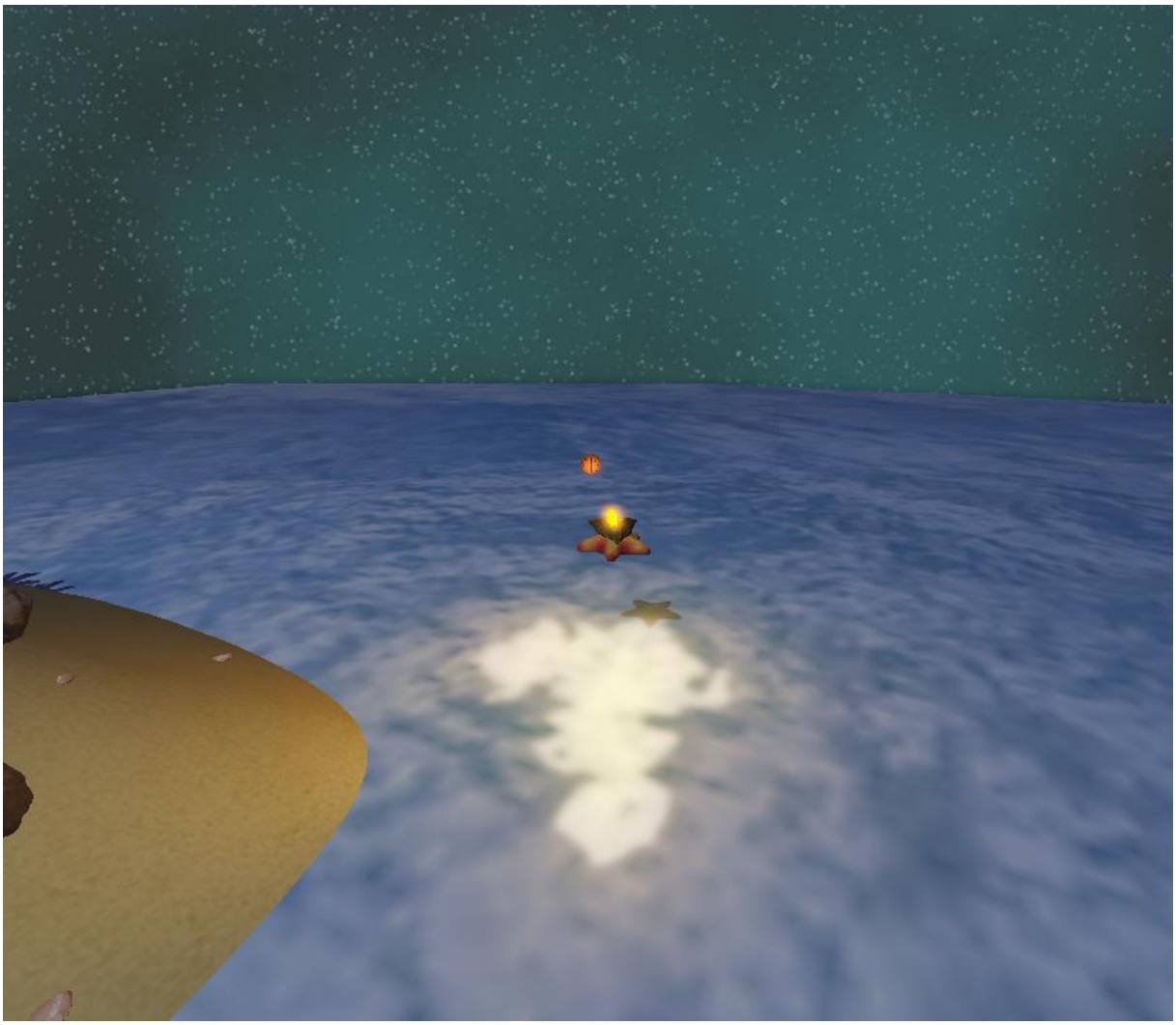


Skákajúci delfín



Kraby chodiace pod kameň

Magická lietajúcasvietiaca hviezdica s orbitujúcou loptou



Pohľad na ohne v lese a odrážajúci sa kokos