Seganku: CGUE 15S Submission 1

Johann Sebastian KIRCHNER (0926076), Nikolaus LEOPOLD (1327344)

Seganku is an animal survival hiding and foraging game inspired by 'shelter'.

The protagonist is a small skunk that has to get ready for hibernation. For that purpose you have to eat a certain amount of food to get nice and fat and then find back to your den where you will be safe. Antagonists are eagles, from which you need to hide in bushes, behind rocks etc. All that running around and searching for food also uses some energy, so you have to eat a lot of roots/berries/etc.

Eagles appear circling in the sky at times (audible bird cry, shadow on ground) and when there is eye contact to protagonist (raycast or simply check if not in hidden place), they attack unless player manages to find cover or uses special stink attack (need to eat certain plant to have this ability).

Controls

The player controls the skunk using 4 directional keys and mouse look (orbiting camera). For testing purposes, a free fly camera can be used alternatively.

Key	**Effect**
W,A,S,D,Q,E	Move player / free fly camera
Shift	Hold down to move more quickly
Mouse movement	Rotate camera
Tab	Toggle camera free fly mode
ESC	Quit game

Libraries

in use: C++11, GLFW 3.0, OpenGL 3.3 core, GLEW 1.10, GLM 0.9.5, Assimp 3.0, FreeImage 3.15 Bullet, FreeType and a sound library will most likely be integrated in the future.

Note that this game is developed cross platform on Linux and Windows, the latter being the main target platform.

Development Status & Implementation Details

For detailled documentation please refer to the generated doxygen html files.

- Game Loop:

  The main game loop and the rendering pipeline structure are layed out in the main function, which calls initialization, update, draw and cleanup functions.
- Transformations:

  Transformation matrices and related interfaces are implemented in the SceneObject class. Geometry, Light and Camera are subtypes of SceneObject. Player is a subtype of Geometry.
- Model loading using Assimp:

  This is implemented in the Geometry class, which traverses the loaded assimp scene to create Geometry objects for each assimp mesh structure. A Geometry object holds a list (hierarchy is not needed and the matrices are applied upon loading) of Surface classes, which store the actual vertex data (positions, indices, normals, uvs), communicates with the VBO and stores pointers to the used textures. Textures are referenced in the Geometry object and reused among its Surfaces.

  Note: we currently use the open standard COLLADA to store the model data. Blender is used for modelling or to prepare existing models and to export to COLLADA.
- Texture loading using FreeImage:

  The Texture class loads image files using FreeImage to generate OpenGL textures, which can then be bound to the active texture unit. Image files should have an 8 bit RGB encoding with no alpha channel. For now, we try to load one diffuse, one specular and one normal map for each Surface using Assimp (see above).
- Shading, Lighting and Materials:

  The Shader class implements loading, compilation and linking and usage of the source files of the GLSL shader stages. Although the architecture allows for flexible shader switching, for now we use an Ubershader approach, in that we use just one shader that provides all needed functionality, since allmost all of our objects are textured. For now a Blinn-Phong illumination model is sufficient for our needs, since it also allows for Lambertian lighting.

  The Light class is needed mostly to have dynamically changing lighting (movement and color transitions) which is needed for the day and night cycles.

  Since we only use diffuse textures mostly (specular and normal should be possible but not needed, since they don't really fit the cartoony style), basic shader uniforms and material properties are sufficient. For now, Materials are simple GLSL structs in the shader, encapsulating the samplers, specular color and shininess exponent for the Blinn-Phong model. There is also a Light struct uniform allowing customizable ambient, diffuse and specular colors and light position.
- Low-poly graphics:

  We try to use a consistent low-poly 'cartoony' style, to avoid the uncanny valley effect. Although this is mostly just relevant for human characters, we still want to make sure to avoid any uncanny appearances.

  Trees, shrubs and carrots are modeled by ourselves. The skunk and eagle are adapted models of a squirrel and a hawk respectively.

Gameplay Description

The gameplay implemented in this submission does not represent the final target, but it serves to give a basic overview and understanding.
A bunch of carrots is randomly placed somewhere on the terrain. The player must find it before they starve, which is after one day and night cycle. When the player find the carrot or starve the game pauses and can be terminated using the [ESC] key.
The eagle currently does not interact with the player, it just circles in the sky. It is possible to hide in the bushes, which is a feature. Hiding in the trees is not a feature though ;) collision is still to be implemented.

Effects

- There is one directional light source representing the sun.
- The illumination model used is simple Blinn-Phong, with customizable GLSL struct uniforms for light and material
- Dynamic day and night cycle (sun movement and color transitions)
- Textured skunk, eagle, carrots, terrain, trees (bark and foliage) and shrubs.
- Consistent low-poly graphics to avoid 'Uncanny Valley' effect.
- Random carrot placement
- The eagle circles the sky in a way that simulates the effect of the wind, simply from an association of translational and rotational components with trigonometric functions.
- Two camera modes: Follow Player (default) and Free Fly. Toggled by [Tab] key. Zoom camera (change FoV) using mouse scroll.

Refer to the next page for ingame screenshots.

Screenshots

for comparison.