Speedgolf!

by Thomas Plagakis for Ubisoft Next 2025

https://github.com/plagakit/ubisoft-next-2025
https://youtu.be/cpkBYS8Tc-o

Speedgolf is like minigolf, but instead of trying to get the fewest strokes, the player tries to get the fastest time. Courses are procedurally generated, and the player has to reach the hole at the end of the course as fast as possible before time runs out. Once they reach it, a new course is generated for the next round and some time is added to their remaining time left. Try and get a high score!

Controls:

- Click and drag to move the golf ball – an arrow will appear where your mouse is showing where the ball will go. Drag it farther to shoot faster.
- T to toggle debug mode

If you see a CRT-like scanline effect in the game, run it in Release mode! In Debug mode, I added that effect to reduce the number of App::DrawLine calls, since those were the main bottleneck causing the game to be unplayably slow. This problem isn't present in Release mode, and the game runs very smoothly there.

The game is built on top of an engine-like static library which wraps over the Ubisoft Next API. It's kinda big (and you probably have a lot to judge), so I've written in the Engine section an overview of the key parts, then in the Game section and overview of Speedgolf!, and then finally in Highlights a few things I'm proud of. If you want to look deeper, each file should have some documentation, such as a class brief.

Engine

The code for the engine can be found under `Engine/`

Folder	Brief

`audio/`	The Audio resource - a wrapper over App's PlaySound.
`components/`	Definitions for plain-old-data containers of game info, typically for use in the ECS
`core/`	Most of the generic backbone-like code of the engine, from app initialization, to debugging, to important services like input or resources
`core/input/`	The engine's input system, which lets programmers define an action and link input events to the action for reusability and flexibility (such as changing keybinds in game)
`core/signal/`	Callbacks similar to std::function, but using templates to bind functions at compile time. Also has Signals – an implementation of the Observer design pattern.
`entity/`	The core of the ECS, defining what an entity is and how they are contiguously stored in their generic memory pools. Also defines EntityViews - fast, direct iterators over the memory pools.
`graphics/`	Everything needed to draw things on screen, from resources (textures, fonts, meshes), to the big renderer
`graphics/renderer/`	Both 2D and 3D rendering code that supports shapes, text, and 3D meshes. Wraps over App's drawing functions. Contains two different types of rasterizers that specialize in drawing different shading modes. See 3D Rendering section for more.
`gui/`	Basic GUI system modelled after ROBLOX's GUI system. Supports hierarchical GUI with buttons, labels, frames, and custom GUI objects.
`math/`	Vectors, matrices, quaternions, and utilities
`physics/`	(So far only) 2D physics body definitions and solvers. Dispatches collisions between two generic bodies using the Visitor design

	pattern.
`scene/`	Simple scene definition, along with storing references to essential game services
`systems/`	Collection of drag-and-drop systems for an Application to use with the ECS. They can make their own System, or use some of the available ones (ex. movement, rendering, etc.)

Project Configuration

- This is a C++17 application because Engine takes advantage of constexpr if, structured binding, std::variant, fold expressions, and <filesystem>
- Engine is a static library that acts as a wrapper over the NextAPI and greatly expands on its features into a full 3D ECS-powered game engine/framework thingie
- Any game using Engine has to statically link to it, as well as dynamically link to the freeglut DLL that App uses
- Games derive from the Application class and use the REGISTER_GAME macro to set up an entry point and make sure everything in the engine gets initialized
- Compile/Linker Definitions: ENABLE_ASSERTIONS, PLATFORM_WINDOWS
- In the future I plan to add functionality for different platforms (Emscripten w/ OpenGL) using the PLATFORM_WINDOWS flag

Entity Component System

- ECS scenes implement an Entity-Component-System framework for managing game objects by using the EntityManager class
- Uses object pooling by storing game objects as entities with components, which are stored in SparseSets fast iteration and no memory fragmentation
- Entities are represented by unsigned integers - UIDs, and are implicitly stored as indices to arrays of components
- Components are POD structs that hold data and are stored in contiguous arrays using SparseSet, all members are public so they aren’t named m_name

- Systems iterates over component arrays and performs operations on them using EntityViews, which are special iterators that iterate directly over the component arrays with no indirection or hashing - fast! :D

3D Renderer

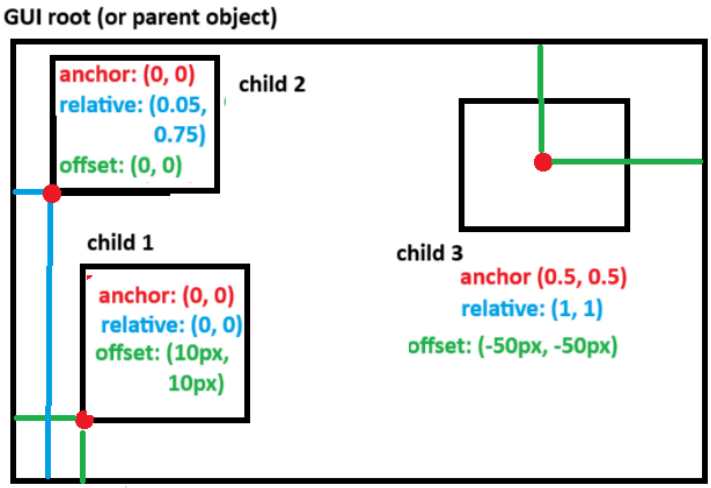
- Left-handed coordinate system, clip space is [0, 1], supports wireframe rendering
- Tries to emulate the functions of a graphics library like OpenGL with draw calls
- Two rasterization modes since the only draw function available is DrawLine:
- Painter's: maintain a list of primitives, sorts them when rendering, and when drawing each primitive calls App::DrawLine on their own. Faster for wireframes.
- Depth Buffer: implement a depth buffer, draw each pixel but clump pixels together into lines to reduce the number of App::DrawLine calls; in the best case we draw only a few lines per row. Faster for filled-in primitives, since pixels are clumped together.

Resource Management

- Instead of passing around shared pointers to resources, they are stored as handles

GUI

- Custom GUI system inspired by how Roblox does it
- Each GUI object stored in a transform-like hierarchy
- Dim2's represent Vec2 (pos, size) that is relative to parent GUI plus a pixel offset



Game

The code for Speedgolf! can be found under `Minigame/`.

Features:

- **3D(?) Golf**

    - The game simulates a 3D environment by processing all movement and collisions in 2D, then updating a 3D view to match it on the X-Z plane. See the Fake3D component and system for more info.
    - A technical problem i ran into was during gameplay, rendering was too slow
        - Since most of what I'm rendering are fake 3d walls and the camera is fixed on the ball, I added a heuristic to calculate the distance from wall to ball, and cull if far enough
    - The golf ball and hole are billboard sprites a la DOOM and those era of games
- **Ever-increasing rounds**
    - The game makes you run a golf course for each round, giving you a limited time to complete it - the goal is to get to the highest round.
    - You get to keep your time left over for the next round.
    - Once you lose, the high score is stored back in main game,
- **Procedural Course Generation**
    - The course is procedurally generated from a template, see `res/courses/course0.txt` for the template, maybe try adding your own course parts!
    - Different course parts are defined in the template, and they are randomly picked to come together to create one long course.
    - The length of the generated course increases with round number, making the game harder as you go along.

- **Obstacles**
    - I only had the time to add two basic obstacles:
    - **Lipping-out:** the phenomena when you hit your golf ball and it goes right on the edge of the hole, but the centripetal force make it circle around and miss - it was very funny implementing this and seeing it in action like a real golf ball.
    - **Moving walls:** simple moving walls, which can be added to parts in the course template

Helpful Resources

My code is scattered with links to helpful resources which I learned a ton from – not all of them are here, but these are probably the most helpful ones:

- Game Engine Architecture (a book from my uni's library)
- Game Programming Gems (another book from my uni's library)
- ECS Back and Forth (https://skypjack.github.io/2019-02-14-ecs-baf-part-1/)
- TheCherno (https://www.youtube.com/@TheCherno)
- Optimizing Software Occlussion Culling https://fgiesen.wordpress.com/2013/02/17/optimizing-sw-occlusion-culling-index/
- ROBLOX & Godot for inspiration
- https://easings.net/
- My past submissions & judge feedback from my 2023 submission :D
- Many, many more