Project S.N.A.P

(**S**ightless **N**avigation **A**nd **P**erception)

Dylan Carlson Dustin Fox Andrew Rose

Sponsor: Dan Schneider

Problem Statement:

Our goal is to create an easily distributable standard testing environment for gathering data and metrics to find the best possible acoustic navigation algorithm.

The Test Bed

- What is the test bed?
  - It is used to try out different configurations for our echolocation algorithm.
- Why is the test bed important?
  - It allows us to find the best configuration to use to navigate an environment.

Software Components

Installation Process

A quick and easy download and installation process for easy distribution.

Main Menu

The main user interface to the testbed.

Logging/Analytics

For each test we need to gather as much data as we can for research.

Configuration

We want to make absolutely everything configurable.

Standard testing maps

A set of standard maps or tests that we can use to test navigation.

Backend

The visual analysis algorithm.

Character Controller/ Headset simulator

A prefab in unity that exactly simulates the functionality of the physical prototype.

2nd Semester Progress

Backend

- New object oriented design
- Combine shared memory access into one DLL
- New build system

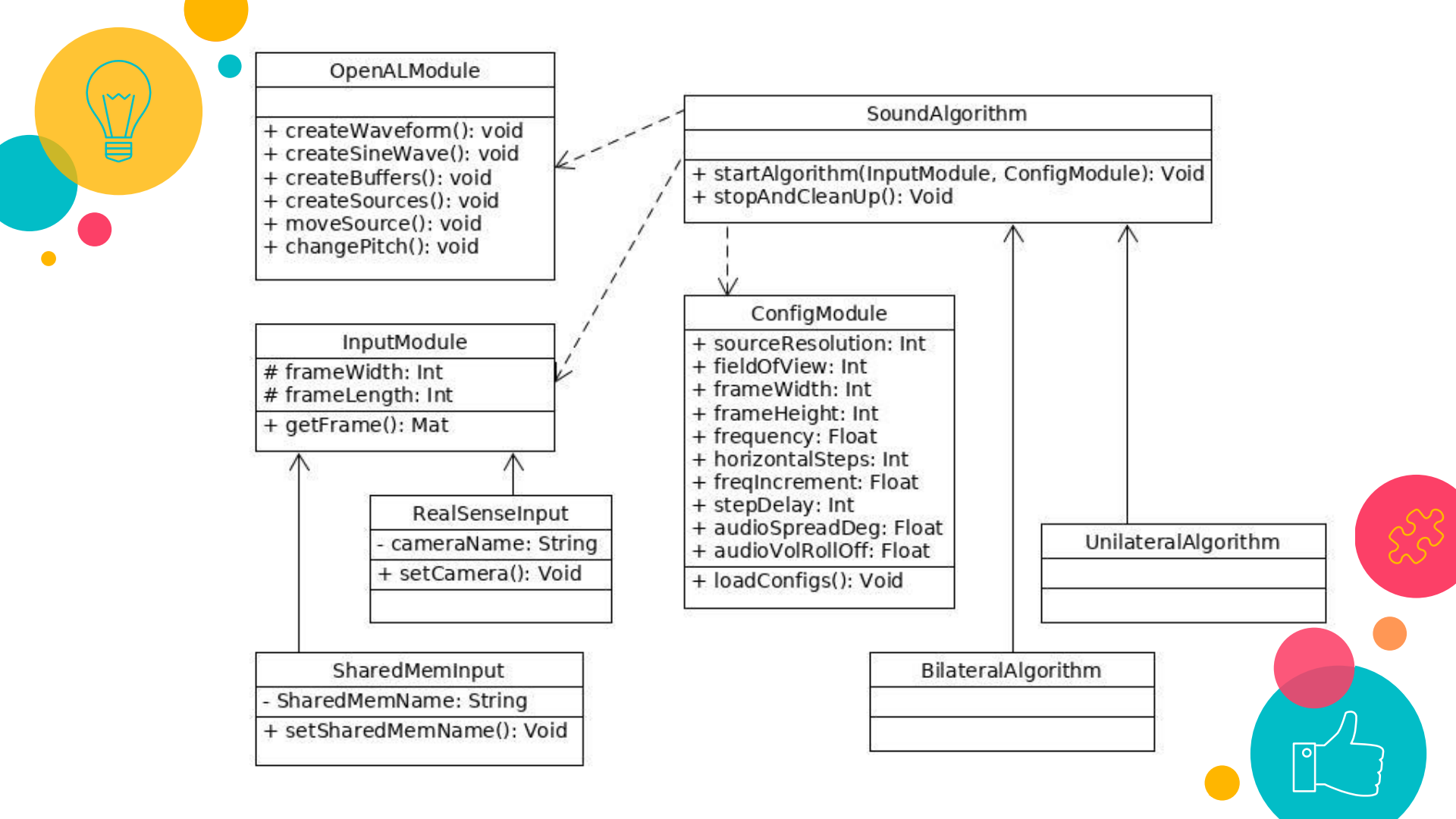
Menu System & Configuration Settings

- Created new main menu
- Added Configuration menu with changeable settings
- Implemented a Save/Load feature for configuration files

Testing Maps

- Two iterations of the hallway map
- Random object generation
- Character controller
- Basic physics

Backend


OpenALModule
+ createWaveform(): void
+ createSineWave(): void
+ createBuffers(): void
+ createSources(): void
+ moveSource(): void
+ changePitch(): void
SoundAlgorithm
+ startAlgorithm(InputModule, ConfigModule): Void
+ stopAndCleanUp(): Void
InputModule
frameWidth: Int
frameLength: Int
+ getFrame(): Mat
ConfigModule
+ sourceResolution: Int
+ fieldOfView: Int
+ frameWidth: Int
+ frameHeight: Int
+ frequency: Float
+ horizontalSteps: Int
+ freqIncrement: Float
+ stepDelay: Int
+ audioSpreadDeg: Float
+ audioVolRollOff: Float
+ loadConfigs(): Void
RealSenseInput
- cameraName: String
+ setCamera(): Void
UnilateralAlgorithm
SharedMemInput
- SharedMemName: String
+ setSharedMemName(): Void
BilateralAlgorithm

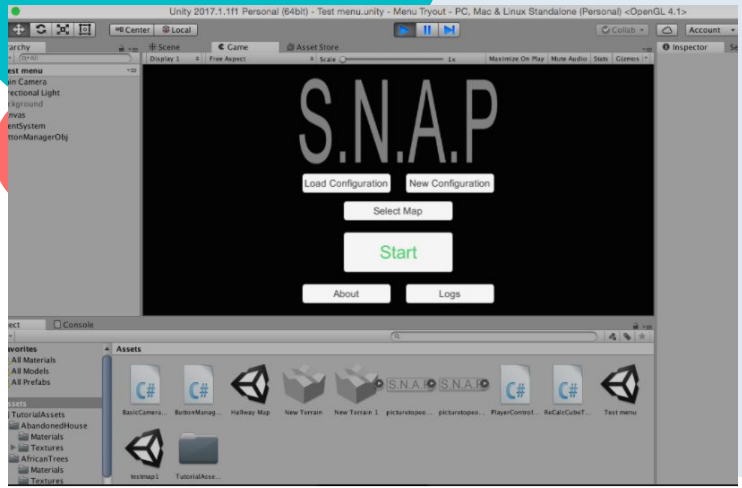
Combined DLL

- One DLL in c++ to facilitate shared memory access for the test-bed
- This allows us to get rid of redundant code
- Makes everything more consistent

Main Menu & Configurations

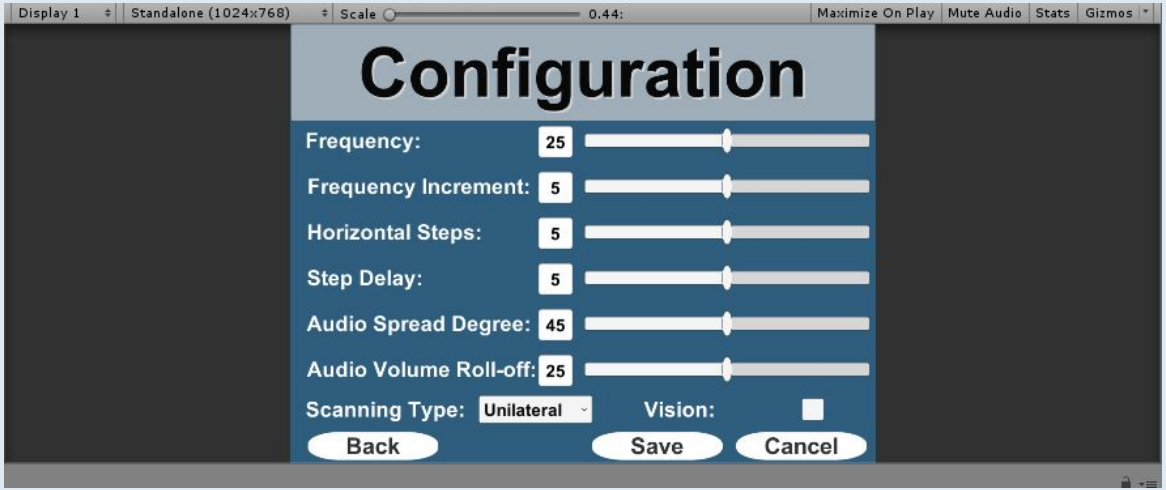
Main Menu User Interface

- Created a new user interface that looks nice and is easy to use
- All components of the test bed can be accessed from the main menu user interface



Configuration Menu

- Configuration menu uses sliders and input fields to easily change settings

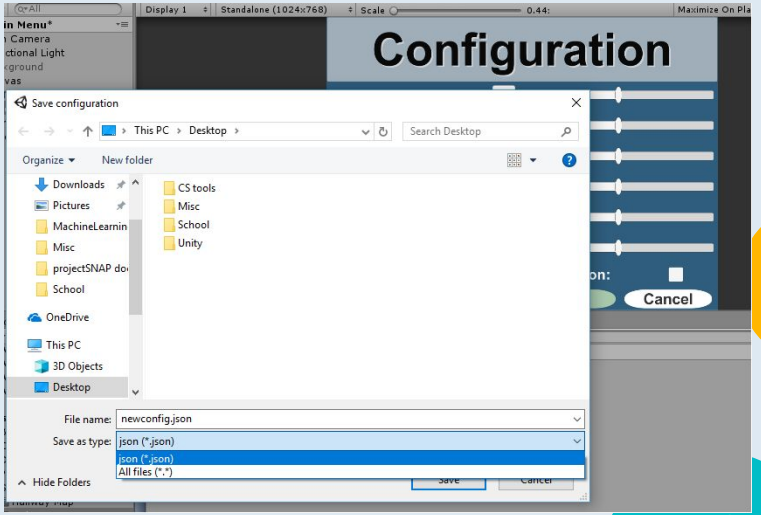


Save/Load Feature

- You can save your configuration settings as a JSON file, which will be read by the Visual Audio Engine

newconfig.json

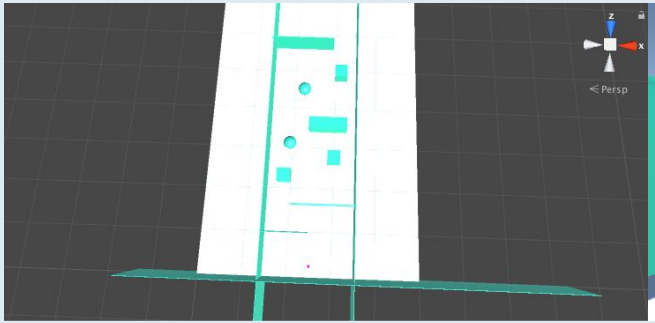
```
{"savedFrequency":25.0,
"savedFrequencyIncrement":5.0,
"savedHorizontalSteps":5.0,
"savedStepDelay":5.0,
"savedAudioSpreadDegree":45.0,
"savedAudioVolumeRollOff":25.0,
"savedScanningType":0,
"savedVision":false}
```

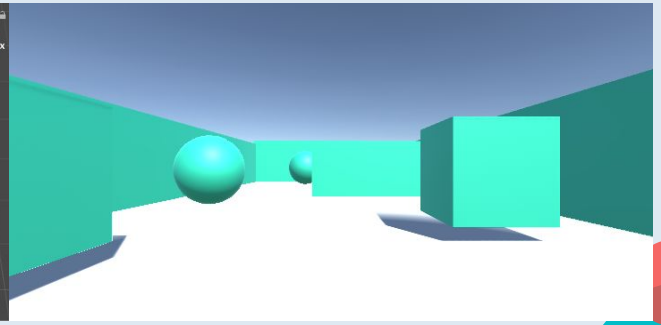


Testing Maps

Hallway Map: First Iteration

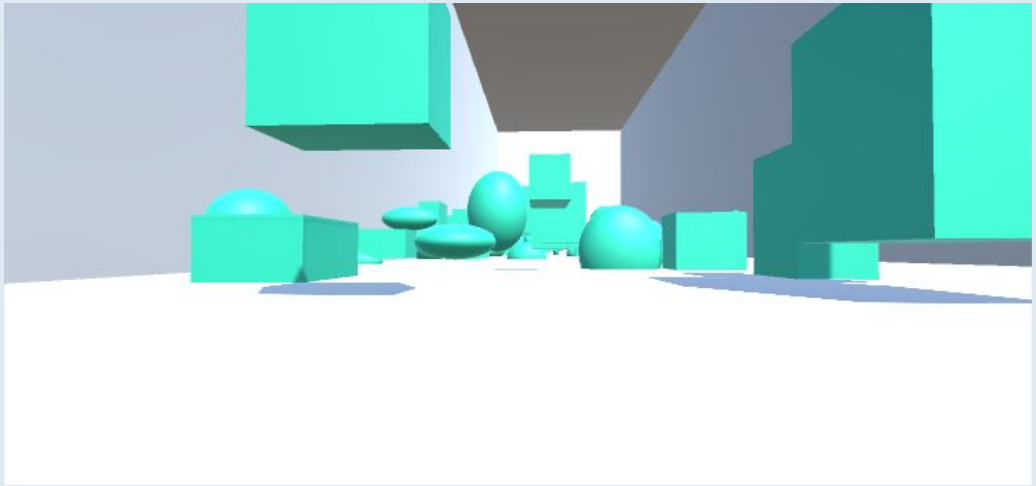
- Our first iteration of the hallway map allowed us to test out map creation, collision detection, physics, and basic character controller functions.
- This iteration was too small and didn't include random obstacle generation.





Hallway Map: Second Iteration

- Larger map with correct scaling
- Implemented random obstacle generation
  - Randomly sized cubes and spheres
  - Easily modifiable



Next Maps

- Dynamic map
  - Dynamic obstacle avoidance
  - Includes random object generation
  - Varying speeds of objects

- Staircase map
  - Incorporates y-axis
  - Random object generation will utilize building/staircase prefabs

Logging

Logging

- Why do we need detailed logging?
  - By recording user information about each test, we can acquire metrics about each audio configuration.
- What will be logged?
  - Object collisions
  - Total map completion time
  - Subjective user review of audio configuration

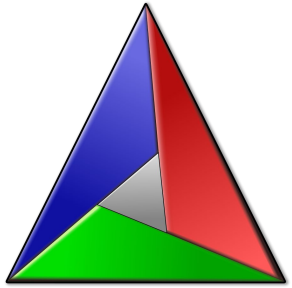
Installation & Distribution

Build System

A single cross platform build process would allow for one codebase to be used with the test-bed as well as any prototypes.

CMake

Cross-platform Make



```
# Set project name, language, and version
cmake_minimum_required(VERSION 3.0 FATAL_ERROR)
project (SNAP-visual-audio-engine VERSION 0.1 LANGUAGES CXX)
SET(CMAKE_SYSTEM_NAME Windows)
# Set build type default to Release
if(NOT CMAKE_BUILD_TYPE)
  set(CMAKE_BUILD_TYPE Release)
endif()

# Get the required packages
find_package(OpenCV REQUIRED)
find_package(OpenAL REQUIRED)
# Set project include directory
include_directories(
       include
       ${OpenCV_INCLUDE_DIRS}
       ${OPENAL_INCLUDE_DIR}
)
# Set project source directory
file(GLOB SOURCES "src/*.cpp")
file(GLOB HEADERS "include/*.h")
# Set the build directory
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR}/build)

# Set the name of the executable
add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})

# Set the libraries to link with the project
target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS} ${OPENAL_LIBRARY})
```

Thanks!

Any questions?