CONTENTS

GAME DESCRIPTION

Soundscape is a game about using sound and touch to orient yourself and become aware of your surroundings. You are dropped into a rectangular stage where you will see yourself, your opponents and an ally (when playing two player mode).

The game is designed to be best played with an Xbox controller; however, keyboard controls were added for testing. It is much more difficult to play via keyboard since you will not have precise control of your sonar and because you won't get the rumble feedback when running into walls / other players.

To find more information about SoundScape have a look at http://soundscape.steelcomputers.com

GAME FEATURES

LOCAL & GLOBAL HIGH SCORES

When you get a high score it is saved locally to a JSON file and globally via a MySQL database at node.steelcomputers.net.

AUDIO AND TACTILE FEEDBACK

Gameplay involves using the volume of your sonar ping in a certain direction. Power of your sonar pulse (the distance it travels) is proportional to the distance you move the joystick.

CONTROL REMAPPING

Each time the game loads; it reads PlayerOneControls.json & PlayerTwoControls.json. Change these to customize controls to something that works better for you.

GAME STORY

Year 2200 now a days we are fighting for the minerals and resources all around the universe.

Now, you are with your childhood friend in the space somewhere close to our solar system and the only detection system you have, is the sonar. So you can only detect alien space ships and your surroundings with it.

The purpose of the game is to identify, survive and kill the enemies with your space ship tools. Using your detection sonar is quite simple, you will send a sonar impulse wherever you are on screen, in a specific direction.

Then by the volume of the sound emitted by the sonar you will get an idea of how close or far are you from an object. The louder, the closer the object is, and vice versa. Keep in mind that is important the intensity of the sonar you apply when searching around you. The higher the intensity is, the louder the response from the sonar will be.

The sonar will detect the other player, walls, enemies and the screen borders.

With the right controller trigger, you can load your gun and shoot it in any direction. Keep in mind to load the gun takes about a second.

You will feel vibrations on the controller when you hit a wall or any screen border.

Make sure you kill them before they kill you! Good Luck!

NAMESPACES

SOUNDSCAPE {

GAMELOOP

The game loop, which inherits from MICROSOFT.XNA.FRAMEWORK.GAME, handles directing the user to different scenes in the game as well as handling the input for the menus. It also provides access to shared resources such as fonts, menu sounds, the sprite batch and some ***GameScenes***

GAMEPLAYSCENE

Gameplay Scene Component abstract class inherits from XNALIB.SCENES.GAMESCENE. It extends the functionality of that class to

TARGETEDGRAPHICSDEVICEMANAGER

This class extends MICROSOFT.XNA.FRAMEWORK.GRAPHICSDEVICEMANAGER but takes a screen number as an argument, allowing uses to play the game on a specified monitor identified via a run argument. IE: "soundscape.exe 1" will run the game on \\Display1 if that display exists.

TOOLBOX

Toolbox is a static method that handles miscellaneous tasks such as saving/loading data to a file or database.

VIRTUALCONTROLLER

This class was a solution to the problem of handling multiple gamepad and keyboard states in a number of different scenes. Before this class we had statements like the following all over the place:

```
if (ks.IsKeyDown(Keys.Escape) && _oldKeyboardState.IsKeyUp(Keys.Escape) ||
        ps.IsButtonDown(Buttons.Back) && _oldPadState.IsButtonUp(Buttons.Back))
```

Not only is this difficult to read, it's only allowing input from a single player's controller. Immediately after creating the ***VirtualController*** class we were able to replace the above with the following:

```
if (PlayerOne.ActionBack || PlayerTwo.ActionBack)
```

That code not only looks much cleaner, it makes it easier to use a lot more buttons to navigate menus, and keeps navigation buttons consistent across all scenes.

For more information on how this class and Lambda expressions cleaned up my code, see "**I <3 Lambda Expressions**" at http://pastebin.com/1MgvGBcw.

SCENES

The following classes all inherit from XNALIB.GAMESCENE. More details on ***GameScene*** can be found on page 6.

STARTSCENE

This scene provides a main menu. It's not that interesting because its 99% a wrapper for ***MenuComponent***.

INFOSCENE

The info scene is used to display a single image to the user. The "Help" and "How To Play" pages are built using this class. Also the ***HighScoreScene*** extends it.

CONTROLLERSTATUSSCENE

This scene is used to display the status of the gamepads overtop of the main menu. It contains a texture to draw when a controller is connected and one to draw when it is disconnected. Player One is drawn in blue while Player Two is drawn in red.

References to the ***GameLoop***'s PlayerOne and PlayerTwo ***VirtualController*** properties are used to poll status information.

HIGHSCORESCENE

This scene is used to display Scores to the user. The scores are displayed over an image so it extends ***InfoScene*** rather than ***GameScene*** directly to reuse the most amount of code.

This scene is only built once and is referenced by other scenes through a property on ***GameLoop*** in order to call helper methods that let other parts of the program know if a score is eligible to be a high score. This way users are only prompted to enter their name when they have a new record.

NEWHIGHSCORESCENE

This scene is called whenever the player score is high enough to get on the Global or Local scoreboards. It consists of 15 ***MenuComponents*** that are positioned in the middle of the screen. When the user hits the select key/button on whitespace or on the last letter position, the score is submitted to the ***HighScoreScene***. After it has been saved, the ***HighScoreScene*** is displayed to the user.

}

SOUNDSCAPE.GAMEPLAYSCENECOMPONENTS {

GAMEPLAYSCENECOMPONENT

This class extends MICROSOFT.XNA.FRAMEWORK.DRAWABLEGAMECOMPONENT. It includes a reference to the scene that it inhabits and has insures that all gameplay classes have the Kill method and the Hitbox and Position public properties. This class also handles drawing so that classes that inherit this are not required to draw themselves (although some do for specific reasons.

This class has three child classes, more information on them can be found below:

WALL

The wall just takes up space and gets collided with. The biggest difference between it and other components is that it's Kill method does not deactivate the object (walls don't die) instead it takes a pre-set number of points away for a "missed shot".

PLAYER

The player object handles storing the channel audio should be played on for that player as well as managing the two rumble motors in the 360 controller. It also handles weapon and sonar logic.

ENEMY

On top of managing its movement, the enemy class handles keeping track of what score a player should get for killing it based on the ***GameplayScene's*** runtime property. Adjustments to score may also be done in the two child classes. More info on Specific enemies can be found below:

CIRCLER

The circler enemy overrides the movement of Enemy so that it moves in a circular pattern. The circler can fly through walls making it very annoying to locate since it will seem to disappear and reappear.

BOUNCER

The bouncer just moves at 45 degree angles, bouncing off of walls in encounters.

}

SOUNDSCAPE.LEVELS {

CAMPAIGN

Campaign is a singleton pattern that makes sure only one level is active at a time and handles providing the levels to the ***GameLoop***.

LEVEL1, LEVEL2, LEVEL3, & LEVELDEBUG

These classes all inherit from ***GameplayScene***. They handle loading textures for the objects that are used in their respective levels. The debug level (***LevelDebug***) is the only level available when the game is compiled using debug mode. In it the enemies do not move, allowing testers to quickly gain access to Victory & Death scenarios and to verify that the score system works as expected.

}

XNALIB {

GAMESCENE

The Game Scene class extends MICROSOFT.XNA.FRAMEWORK.DRAWABLEGAMECOMPONANT, to handle drawing and updating all components within a scene. It will also draw a background image if it has been provided with one through the protected field `_background`.

MENUCOMPONENT

The menu component is a helper class for creating menus. It takes an array of strings for the menu items and can store and object to go along with each item. For example in the ***StartScene*** it stores a ***GameScene*** for each menu item that goes directly to one. For the ***NewHighscoreScene*** class it is used to store each letter in the alphabet as a char. In the gametype submenu it stores a ***GameOptions*** enum.

MENUITEM

A menu item is a structure used by menu component that has a String “Name” and a GenericType Component that can store any kind of object.

}

CLASS DIAGRAM