Voice Activated Control for 2D platformer in Unity 3D

This document describes the details of addition of voice activated control support in the game 2D Platformer. The game is available on the Unity 3D platform and normally supports keyboard controls.

Basic player moves available in the game are as follows:

- Player moves in a 2D side scrolling plane, either from left to right, or from right to left.
- Player can jump straight up when grounded.
- Player can shoot and kill enemies with a gun.

Voice control support has been added for these moves. When the player 'hears' the 'command' from the user, he is required to perform the requested action. The term 'hearing' in this scenario refers to the interception of human voice signal through the default audio input device (microphone) present on the computer where the game is being played. The details of the voice 'commands' and their intended results are as follows:

1. Jump
   On 'hearing' the keyword "JUMP" , the player jumps up from the reference ground plane.
2. Right
   On 'hearing' the keyword "RIGHT",the player moves towards a certain distance right from his current position.
3. Left
   On 'hearing' the keyword "LEFT" ,the player moves towards a certain distance left from his current position.
4. Shoot
   On 'hearing' the keyword "SHOOT" ,the player pulls out his gun and shoots a bullet.

Assumptions for proper functioning of the voice control system:

- Only one 'command' is processed at a particular time. Until the result of the previous command is completed, the next command will not be processed.
- The player must be in its stationary state before it can execute any 'command'.
- 'Commands' are recognized individually only. Combinations (Overlapping) of two or more commands would not result in desired player movement.
- No disturbance in the input in form of environmental noise may be present.
- The user can speak the mentioned 'commands' in the English language. Only the 'commands' mentioned above will result in player actions. Any other input may be ignored.

Implementation Details

The voice recognition application has been implemented in C# using Microsoft Speech.Recognition APIs. The recognition system runs as a separate process on the computer. The game communicates asynchronously with the system via a local network interface. The

game passes on the voice input to the system which then processes and returns the appropriate recognized text commands to the game. The game has a dedicated controller thread to receive these text commands and appropriately begin modification of the game state.

Running the computationally intensive processing as a separate thread allows the game to remain responsive while the computation is done in the background. Also running the recognition engine as a separate process allows independent development and modification of the recognition engine without frequent changes to the game.