Survival Shouter Functional Specifications

Gameplay

Survival Shouter modifies the top-down shooter Unity game “Survival Shooter” by adding voice-controlled bombs. When a command is recognized, a bomb is immediately spawned at the player’s current position and explodes after **1.5 seconds**. Bombs have a cooldown of **10 seconds** and damage both enemies as well as the player within a blast radius of **2.25 meters**. The danger to the player is meant to balance the increased power given to the player, while the cooldown period prevents spamming.

There is an initial delay of **2 seconds** at the beginning of the game before the user can set a bomb, to prevent accidental activation and sudden death. The bottom right corner of the screen shows the status of bombs: “**Bombs loading**”, “**Bomb ready**”**,** “**Bombs reloading**”, or “**Speech system error**” if the speech recognition backend failed to initialize. The statuses are **white** except for “**Bombs reloading**”, which is **red**. The color alteration provides an easily processed visual cue to the user.

Voice Commands

Any one of a number of voice commands can be issued by the player to set a bomb. It is assumed that the speech recognizer is not always accurate, and therefore the voice activation criteria is flexible. Any recognized phrase with an important keyword present, such as “**bomb**”, “**blow**”, or “**boom**”, is accepted as an activating command.

Speech Recognition API

A native library (splistener) is implemented to provide a simple interface for the Unity game to obtain data from the speech engine backend. splistener takes parameters for language files and keywords. This allows the game to be easily modified to recognize any language and any set of keywords. The backend speech recognition framework pocketsphinx was chosen because it is compatible with all major desktop and mobile operating systems. This makes splistener highly reusable for other applications.

Continuous and Non-Intrusive Speech Decoding

After the Unity game calls splistener’s initialization function, a separate thread is created for initializing the microphone, recording audio, and decoding the audio into speech text. Because the Unity game and the speech recognition run on separate threads after this point, the player can continue playing uninterrupted even if there is a significant slowdown or crash in the speech recognition and audio processing subsystems. The game thread queries splistener for new speech data once every frame. If new data is found, it is parsed for the keywords. Otherwise, the game thread continues.

Improvements

The splistener library currently assumes that only one thread of speech recognition will be initialized at a time by a host process. Therefore, it is not designed to be reentrant. Future updates should implement re-entrancy for safety and can even add the capability of having multiple threads process parallel microphones.