2015

# Technical Design

SPOKEN

DADIU

# Contents



# Figures

## I. The Game

Spoken is a co-op puzzle adventure game where the protagonist along with his/her friend (a sidekick AI) need to go through all the rooms and solve the puzzles there in order to complete the game. Each level in the game represents one room. A player cannot progress further unless the all puzzles are completed. Throughout the game the player explores and learns an artificial language which is used to interact with the AI sidekick. Each time a player learns new sign it is stored in his stone tablet, where the player can go and see/use all signs s/he has learnt.

## II. Standards

- Methodology – Scrum
- Naming format – camelCase
- Code Style - Generic over Hardcoded
- Platform - Android 5, Nexus 9

## III. Technical Features

### 1. Camera

The camera in the game is set to show parallel projection of the player character and it is a free moving camera. This means that the camera can be dragged around by the player to view different parts of the level. However, there are restrictions on how far away from the level the camera can go. The camera can be also rotated and zoomed in/out with the traditional 2-finger gestures. The default camera rotation setting in Unity are (x: 35, y: 35, z: 0) where the z axis is always locked.

### 2. Languages Handling

All syllables, signs, and sentences are stored in separate XML data files. When the game is started, the game reads the content of these files and then files are accessed from the game only if there is any change coming from the player (i.e. create/modify/delete signs). The structures of the XML files are as it follows:

a) Syllables - ID, Image, Sound
b) Signs - ID, Name, Syllable IDs Sequence
c) Sentences - ID, Sign ID Sequence

When the signs are created, the player may choose to combine between 2 and 4 syllables to create a sign and a sequence of syllables is saved to the database files. The images of the syllables are put one on top of the other to form the final sign. Also, when a sign is created, there is sequence from sounds that is saved, as well (the sequence of sounds is just an array that point to different sound files in specific order). The elements in this sequence represent different sounds of the sign and if a sign is pronounced, then all the sounds in the sequence are played one-after-another.

3. AI

The AI in Spoken is built to contain the following properties:

a) States

The AI's movement and actions are handled by a state machine. The AI switches between different states depending on predefined behavior. Examples of the states are the following:

- Following the Player state
- Go to Object And Interact state
- Wandering/Exploring state
- Go to Player and Stay Still state
- Custom state (e.g. a puzzle-specific behavior or extension of an existing state )

The Ai is implemented using the unity co-routine that given a current states executes that state. The current state is implemented using an interface that all states need to follow. The main function is execute state, that is then called on the active state each 0.1 seconds. The general behavior of the sate machine, is like any other, to execute a given state, until that state is done, and then return the default state.

This gives the sensation that the Ai is actually getting bored, if you don't give it any new state, it starts to stroll around the map, and starts to explore on its own.

Most of the states are implemented in a very general way, to make sure we don't need to write the same code multiple times. Some states have couple of sub-states - an example of this is 'GoAndInteractWithNearest' which has the sub-states: Neutral, GoToInteractable, Interact, WaitSomeTime, Done. This means that it is very easy to control and generalize every state, and easily add on new states.

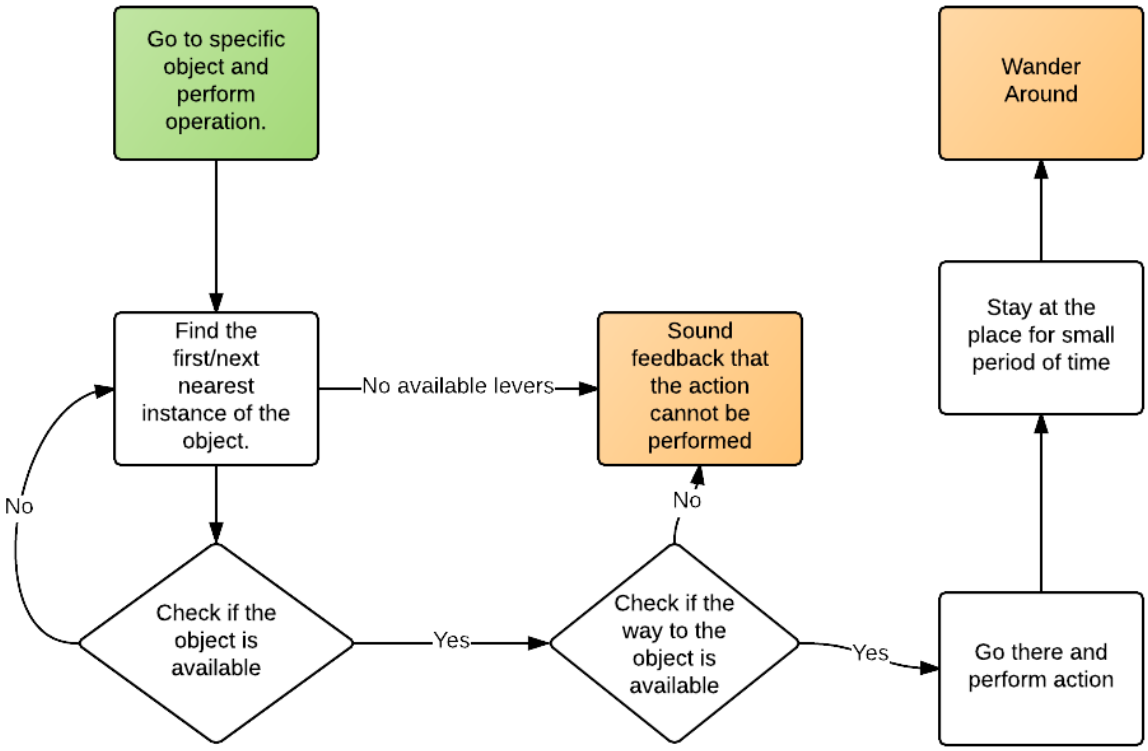
b) Speaking and Understanding User Input (typed signs)

The AI utilizes the database of artificial language signs and uses the IDs of the signs rather than actual signs to speak and understand user's input (i.e. the sentence "you go there" typed with signs will be interpreted as "2, 7, 5"). However, the AI is not advanced enough to work with individual signs (indexes). Therefore, there is a database of predefined word sequences (sentences), and there is an action/reaction implemented for each sentence. If the AI is input a sequence that is not in the database, then there is a default response that the AI communicates to the player.

c) Decision Tree

The decision tree determines how the AI is going to behave when it is instructed something - i.e., if the player commands the AI to pull a lever, the AI would check if the nearest lever is available (the player is not standing near it) and if it is, it will go there, else the AI will go to the other lever. The diagram below shows the full decision tree for the AI.

*Figure 1 - AI decision Tree*




d) Performing Commands

Performing commands is a combination of understanding speech, and performing an action on target object with regard to its decision tree.

e) Following and Pathfinding

For the movement, a standard navmesh is used to calculate the shortest path to an object. Both characters are using the same method to move around. The characters are supposed to stop before the target object and not on it. For this to work, we have added interact point to most of the interactables, to make sure the player does not go into then object.

f) Puzzle-Specific Behavior

There is possibility for the Level designer (with help from a programmer) to add puzzle specific behavior to the AI. A custom behavior can be created by creating a script that implements the IState interface and then it is set to the AI. Furthermore we have created a waypoint system that makes it possible for the level designer alone to do most of the work.

g) Ability to Carry Items

All items that the player and the AI are supposed to carry are going to be part of their prefabs, but they are going to be disabled. Whenever the player picks up an item, it is going to be enabled.

4. In-Game Shop

The in-game shop is a mock-up of a real shop - you can buy syllables packs that will be available the next time when you need to create a sign. All the packs in the shop are bought with soft currency, and when you try to buy soft currency with hard currency, it would not request any additional operations, meaning you can buy as much soft currency as you want.

As a part of the monetization strategy, the player can also unlock new levels using soft currency. However, the new levels are not unlocked through the shop, but rather directly in the level selection screen.

5. Achievements – Discarded / Not Implemented

The achievements in the game are going to be linked to google play achievements through the Google API for Google play achievements.

6. Sounds

The sounds in the game are divided in three streams - Music, Voices and SFX (sound effects). Each stream can be enabled/disabled, and volume can be specified. There is also a soundscape in which random sounds are played and there is a cross-fade effect between them.

7. The UI

- Start screen
- Level selection screen – Select levels to play. Only the unlocked levels can be chosen.
- Settings screen – regulate the sound level for the master stream and its sub-streams or turn them on/off
- Credits screen

- In-game UI
    - Pause menu > Settings
    - Pause menu > Level Selection
- Stone Tablet – this is the place where functionalities that are directly involved with the gameplay and player experience are located.
    - Symbols sub-window
    - Shop sub-window
- End level screen – appears when the level is completed. It displays the following:
    - Which level is completed
    - Orbs collected from the current level
    - Total amount of collected orbs
    - Button that redirects to the level selection screen
    - Button that redirects to the next level

8. Collectibles (Orbs – the in-game currency)

The orbs are created inside Unity using entirely particles with transitions in colors depending on the particle lifetime.

9. UI Messages

The UI messages are implemented in a very easy-to-use and generic way, where the only thing you need to do is call a method and pass it a number parameters depending on which type of window you are invoking (f.x. a "*Yes/No*" window would require title, message, buttons' names, buttons' functionalities), in order to have a message on the screen. So, when a message appears, the game is paused (the timescale is set to 0). When the message is closed, no matter what button has been pressed, the game resumes. In case the window is used for error messages, there should be no OK button and the player should close the app and reinstall it.