Song Match Documentation

The Cozmonauts

May 15, 2018

Getting Started

Welcome to the documentation for the Cozmo Song Match project!

Song Match is a game where you try to match the notes of a song by tapping blocks with Cozmo. With each round, the game gets longer and a little harder. Song Match supports up to 3 players.

Song Match is brought to you by four undergraduate students from the University of Missouri - St. Louis. Collectively, we are The Cozmonauts.

CHAPTER 1

How It Works

The game starts by playing the first three notes of a song. Each time a note is played, a corresponding cube flashes. The player must match the notes by tapping the correct sequence of cubes. If the player gets the sequence correct, then Cozmo tries to match the correct sequence. If either the player or Cozmo gets *three* notes incorrect, then they lose the game.

This makes up **one** round. Each round the length of the sequence increases, until you reach the end of the song.

Please view our User's Guide for detailed information on how to play the game.

`Download User's Guide`

Also, for some ideas on how to pitch the game to children, check out our Educator's Guide.

`Download Educator's Guide`

1.1 Setup

1. Setup the Cozmo SDK.
   - Please see Initial Setup under the Cozmo SDK documentation.
2. Clone the repository.

```
$ git clone https://github.com/gbroques/cozmo-song-match.git
```

3. Navigate to the repository.

```
$ cd cozmo-song-match
```

4. Install dependencies.

```
$ pip install -r requirements.txt
```

Tip: We recommend installing dependencies within a virtual environment. See Virtual Environments for details.

5. With Cozmo connected, run the main program. For details on connecting Cozmo, see Getting Started With the Cozmo SDK.

```
$ python main.py
```

We support command line arguments for configuring what song you want to play, and how many players. Use the help flag `-h` for help.

```
$ python main.py -h

usage: main.py [-h] [-s S] [-p N]

Play Song Match with Cozmo.

optional arguments:
  -h, --help  show this help message and exit
  -s S        The song to play. Hot Cross Buns (hcb), Mary Had A Little Lamb
              (mhall), or Rain Rain Go Away (rrga). Defaults to a random song.
  -p N        The number of players for the game. Defaults to None. If None
              then selecting the number of players will be handled in game.
```

6. Download and Install PyCharm (Optional)

- See why we recommend you develop with PyCharm

1.2 Virtual Environments

The following document addresses what a virtual environment is, and how to set one up.

1.2.1 What Is a Virtual Environment?

To explain what a virtual environment is, and why it's needed, consider the following example:

You have two projects, **A** and **B**. Both depend on different versions of the same library.

- Project A requires version **1.12.4**.
- And Project B requires version **2.5.6**.

How do you manage this?

The solution is to create an isolated *virtual environment* for each project.

`virtualenv` is a tool to create isolated Python environments.

1.2.2 How to Setup a Virtual Environment

The following are steps to setup a virtual environment for the Song Match project.

1. Install virtualenv
2. Navigate to the root of the Song Match repository on your machine.

```
$ cd path/to/song/match/repo
```

Note: See Setup for how to clone the repository.

3. Create the virtual environment. We'll call it `venv`, but you can call it anything.

```
$ virtualenv venv
```

4. Activate the virtual environment. This step depends upon your operating system. See activate script for details.

For **POSIX** based systems:

```
$ source venv/bin/activate
```

For **Windows**:

```
> venv\Scripts\activate
```

5. Install the project's requirements.

```
$ pip install -r requirements.txt
```

6. You can deactivate the virtual environment at anytime with the command:

```
$ deactivate
```

Error: Having Trouble? See virtualenv's documentation for help.

1.3 PyCharm

This document aims to explain why we recommend using PyCharm to develop Song Match.

If you haven't already, download and install PyCharm Community Edition here.

1.3.1 Finding Usages

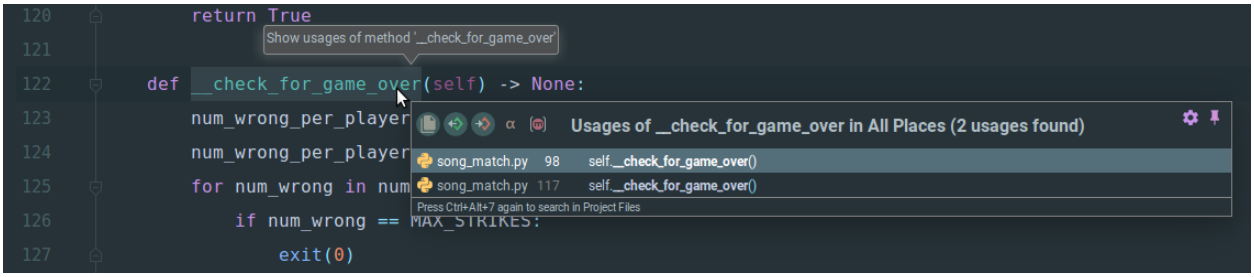
The first reason we recommend you use PyCharm is the ability to find usages. Often when developing you need to find all the places where a function is called. PyCharm makes this easy.

For example, here's a function that checks for whether the game is over in `song_match.py`.

```
120         return True
121
122     def __check_for_game_over(self) -> None:
123         num_wrong_per_player = [player.num_wrong for player in self._players]
124         num_wrong_per_player.append(self._song_robot.num_wrong)
125         for num_wrong in num_wrong_per_player:
126             if num_wrong == MAX_STRIKES:
127                 exit(0)
```

If you hover over the name of the function while holding `Ctrl`, and click the name of the function you can find usages.

PyCharm also has other keyboard shortcuts and ways to find usages that are helpful to learn.



The tooltip window shows the function is called in two places:

1. Line `98` in `song_match.py`
2. and line `117` in `song_match.py`

You can select either usage and PyCharm will automatically navigate there for you.

Note: Finding usages works for functions, methods, variables, and anything else you could care about.

1.3.2 Enforcing PEP 8

PyCharm also helps developers follow Python's official style guide PEP 8.

For example, PEP 8 states variables should use snake case instead of camel case.

PyCharm underlines the camelcase variable with a yellow squiggly line and suggests renaming it from `numPlayers` to `num_players`.

1.3.3 So Much More

PyCharm offers many other great features that are outside the scope of this document like:

- Integration with `requirements.txt` files
- Powerful refactoring abilities
- Creating and Optimizing imports
- Reformatting source code
- and so much more.

You can find more information on JetBrain's website.

1.4 Codebase Conventions

This document aims to explain some of the common conventions found within the Song Match codebase.

1.4.1 Design Patterns

The Song Match codebase is almost entirely object oriented. There are many design patterns relating to object oriented programming (OOP) found throughout the codebase.

Wrapper Objects

The first design pattern you'll see are our use of wrapper objects. A "wrapper object" is an object that takes an already instantiated object in it's constructor and extends it's functionality through custom methods.

More formally this is known as the Decorator Pattern because your *decorating* an object by wrapping it and adding behavior.

Common objects from the Cozmo SDK, like the objects representing Cozmo and the cubes, have a corresponding wrapper object in the Song Match codebase:

- *SongRobot* wraps `Robot`
- *NoteCube* wraps `LightCube`
- *NoteCubes* wrap a list of 3 `LightCube` instances

Many methods and properties of the wrapper objects match the corresponding wrapped object. For example, both *NoteCube* and `LightCube` have a `set_lights` method. The *set_lights()* method in `NoteCube` simply calls `set_lights()` on the internal `LightCube` object.

```
# note_cube.py

def set_lights(self, light: Light):
    self._cube.set_lights(light)
```

Object Creation Patterns

Static Factory Methods

A *static factory method* is a static method used for creating an object.

There are two *static factory methods* in the codebase. Both are methods named `of`, a concise naming convention for static factory methods popularized by `EnumSet` in Java. See How to name factory like methods? for more details.

- `NoteCube` - *of()*
- `NoteCubes` - *of()*

Factories

A factory is an object for creating other objects.

In our codebase there is one factory, *EffectFactory* .

EffectFactory creates our various game effect subclasses:

- *CorrectSequenceEffect* - Played when a player matches the correct notes.
- *RoundTransitionEffect* - Played when transitioning between game rounds.
- *WrongNoteEffect* - Played when a player fails to match the correct notes.

1.4.2 Inheritance

We favor composition over inheritance and avoid complex class hierarchies.

No class extends an instantiable class, but there are two abstract base classes:

- *Effect* - Abstract base class for various game effects
- *Song* - Abstract base class for various songs

1.4.3 Public, Protected, and Private

Python lacks access modifiers like `private` and `protected` found in languages like Java and C#.

We follow the convention of preceding `private` methods and attributes with two underscores. For example:

```
def __some_private_method():
    pass
```

`protected` methods and attributes are preceded with a single underscore.

```
def _some_protected_method():
    pass
```

If you see anything that begins with a underscore, then it means don't use it outside of that class or module.

In general, all `public` members in Song Match have docstring comments, while `private` and `protected` members do not.

Our API reference includes only `public` members.

1.4.4 Type Hinting

In general, all functions and methods are type hinted. Below we see a function that adds two `int` values together, and returns an `int`.

```
def add_two_numbers(a: int, b: int) -> int:
    return a + b
```

See support for type hints for more details.

1.4.5 Style Guide

We follow Python's official style guide PEP 8.

1.5 Package Structure

This document aims to give a high-level overview of the Song Match package structure.

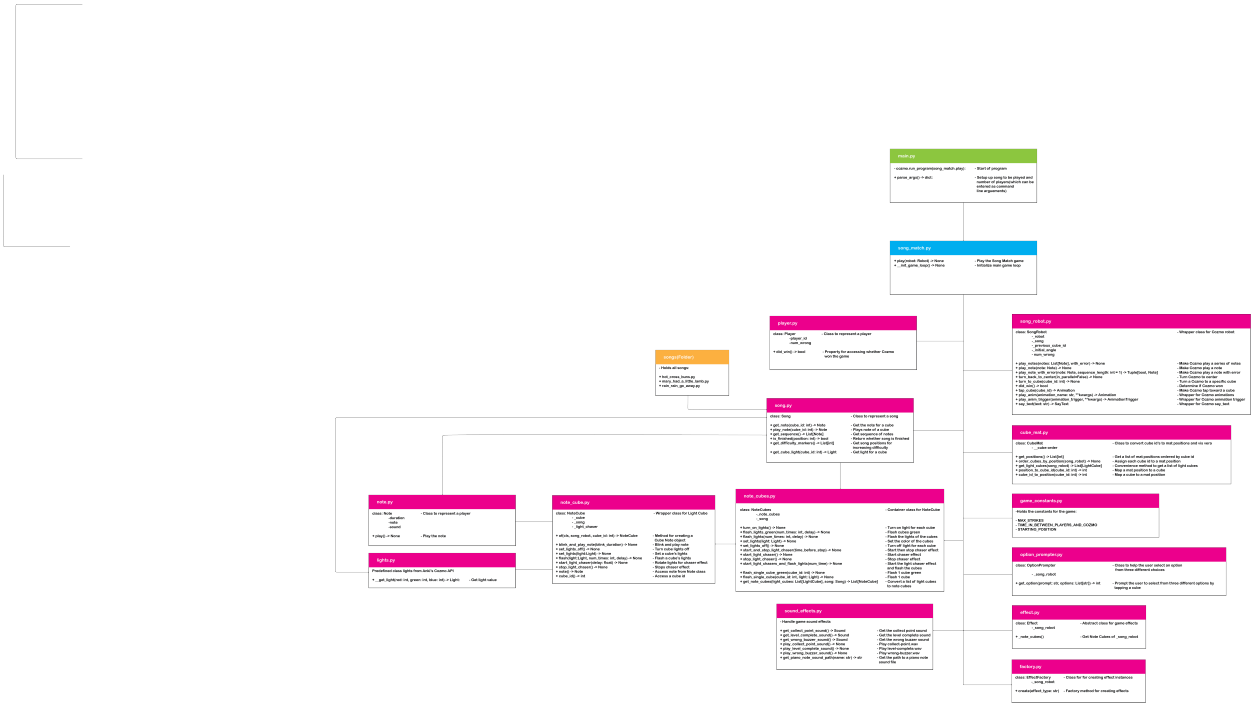
- `song_match` - Contains most of the code.
  - `cube` - Code related to Cozmo's cubes.
  - `effect` - Code related to various game effects.
  - `exceptions` - Custom exceptions.
  - `sound_effects` - Sound effects and `.wav` files.
  - `song` - Code related to songs.
- `test` - Contains unit tests.
- `docs` - Contains the documentation.

For more detail, dive into our API reference.

1.6 Game Flow

Here is a visual on how the program is structured. It can also be downloaded using the link below.

`Download Song Match Program Flow`



1.7 song_match

- *SongMatch* - Main game class.
- *SongRobot* - A wrapper around Robot.

1.7.1 song_match.cube

- *NoteCube* - A wrapper around LightCube.
- *NoteCubes* - Container class for three *NoteCube* instances.

song_match.cube.lights

Predefined Light instances for Cozmo's cubes.

song_match.cube.lights.**BLUE_LIGHT = <cozmo.lights.Light object>**
Light - Blue light instance.

song_match.cube.lights.**CYAN_LIGHT = <cozmo.lights.Light object>**
Light - Cyan light instance.

song_match.cube.lights.**GREEN_LIGHT = <cozmo.lights.Light object>**
Light - Green light instance.

song_match.cube.lights.**ORANGE_LIGHT = <cozmo.lights.Light object>**
Light - Orange light instance.

song_match.cube.lights.**PINK_LIGHT = <cozmo.lights.Light object>**
Light - Pink light instance.

song_match.cube.lights.**RED_LIGHT = <cozmo.lights.Light object>**
Light - Red light instance.

song_match.cube.lights.**YELLOW_LIGHT = <cozmo.lights.Light object>**
Light - Yellow light instance.

song_match.cube.note_cube

Module containing *NoteCube*.

class song_match.cube.note_cube.**NoteCube**(*cube: cozmo.objects.LightCube*, *song: song_match.song.song.Song*)

Bases: object

Wrapper class for a LightCube to play a note when tapped.

blink_and_play_note(*blink_duration=0.125*) → None
Blink the cube and play the corresponding note.

Parameters **blink_duration** – How long the cube blinks for in seconds.

Returns None

cube_id
Property to access the cube_id of a cube.

Returns cube_id

flash(*light: cozmo.lights.Light*, *num_times: int*, *delay=0.15*) → None
Flash a light a certain number of times.

Parameters

- **light** – The light to flash.
- **num_times** – The number of times to flash the light.
- **delay** – Time in seconds between turning the light on and off.

Returns None

note
Property to access the *Note* of a cube.

Returns *Note*

classmethod of(*song_robot*, *cube_id: int*) → song_match.cube.note_cube.NoteCube
Static factory method for creating a *NoteCube* from *SongRobot*.

Parameters

- **song_robot** – *SongRobot*
- **cube_id** – cube_id

set_lights(*light: cozmo.lights.Light*) → None
Wrapper method for set_lights().

Returns None

set_lights_off() → None
Wrapper method for set_lights_off().

Returns None

start_light_chaser(*delay: float = 0.1*) → None
Rotates the cube's color around the light corners in a continuous loop.

Parameters **delay** – Time awaited before moving the rotating lights.

stop_light_chaser() → None
Ends the light chaser effect.

Returns None

turn_on_light() → None
Turn on the light for the cube assigned in *Song*.

Returns None

song_match.cube.note_cubes

Module containing *NoteCubes*.

class song_match.cube.note_cubes.**NoteCubes**(*note_cubes: List[song_match.cube.note_cube.NoteCube]*, *song: song_match.song.song.Song*)

Bases: object

Container class for three *NoteCube*.

flash_lights(*num_times: int = 4*, *delay=0.15*) → None
Flash the lights of each cube.

Parameters

- **num_times** – The number of times to flash lights.
- **delay** – Time in seconds between turning the light on and off.

Returns None

flash_lights_green(*num_times: int = 3*, *delay=0.15*) → None
Flash the lights of each cube green.

Parameters

- **num_times** – The number of times to flash green.
- **delay** – Time in seconds between turning the light on and off.

Returns None

flash_single_cube(*cube_id: int*, *light: cozmo.lights.Light*) → None
Flashes the light of a single cube, while turning the lights of the other cubes off.

Parameters

- **cube_id** – `cube_id`
- **light** – The `Light` to flash.

Returns None

flash_single_cube_green(*cube_id: int*) → None
Convenience method for calling *`flash_single_cube()`* with a `green_light`.

Parameters **cube_id** – `cube_id`

Returns None

flash_single_cube_red(*cube_id: int*) → None
Convenience method for calling *`flash_single_cube()`* with a `red_light`.

Parameters **cube_id** – `cube_id`

Returns None

static of() → song_match.cube.note_cubes.NoteCubes
Static factory method for creating *`NoteCubes`* from *`SongRobot`*.

Parameters **song_robot** – *`SongRobot`*

set_lights(*light: cozmo.lights.Light*) → None
Call `set_lights()` for each cube.

Parameters **light** – `Light`

Returns None

set_lights_off() → None
Call `set_lights_off()` for each cube.

Returns None

start_and_stop_light_chasers(*time_before_stop=2*) → None
Starts and stops the light chaser effect for each cube.

Parameters **time_before_stop** – Time to wait before the light chaser effect stops (in seconds).

Returns None

start_light_chasers() → None
Starts the light chaser effect for each cube.

Returns None

start_light_chasers_and_flash_lights(*num_times=2*) → None
Starts the light chaser effect and flashes the cubes

Parameters **num_times** – The number of times to perform the light chaser effect and flash the cubes.

Returns None

stop_light_chasers() → None
Stops the light chaser effect for each cube.

Returns None

turn_on_lights() → None
Turn on the light for each note cube.

This method turns on the lights assigned in *Song*.

Returns None

song_match.cube.note_cubes.**get_note_cubes**(*light_cubes: List[cozmo.objects.LightCube], song: song_match.song.song.Song*) → List[song_match.cube.note_cube.NoteCube]
Convert a list of light cubes to note cubes.

Parameters

- **light_cubes** – A list of three LightCube instances.
- **song** – *Song*

Returns A list of three *NoteCube* instances.

song_match.cube.util

song_match.cube.util.**get_light_cube**(*song_robot*, *cube_id: int*) → cozmo.objects.LightCube
Convenience method to get a light cube instance.

Parameters

- **song_robot** – *SongRobot*
- **cube_id** – cube_id

Returns LightCube

song_match.cube.util.**get_light_cubes**(*song_robot*) → List[cozmo.objects.LightCube]
Convenience method to get a list of light cubes.

Parameters **song_robot** – *SongRobot*

Returns A list of three LightCube instances.

1.7.2 song_match.effect

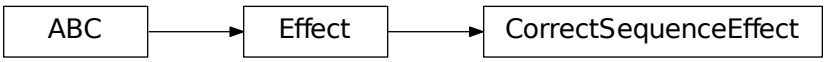
- *Effect* - Abstract base class for game effects.
- *EffectFactory* - Factory for creating *Effect* instances.

song_match.effect.effects

Package containing various game effects.

- *CorrectSequenceEffect* - Played when a player matches the correct notes.
- *RoundTransitionEffect* - Played when transitioning between game rounds.
- *WrongNoteEffect* - Played when a player fails to match the correct notes.

song_match.effect.effects.correct_sequence



class song_match.effect.effects.correct_sequence.**CorrectSequenceEffect**(*song_robot: song_match.song_robot.Song*

Bases: *song_match.effect.effect.Effect*

Played when either a player or Cozmo matches a sequence of notes correctly.

play(*is_sequence_long: bool = False*, *is_player: bool = True*) → None
Play the correct sequence effect.

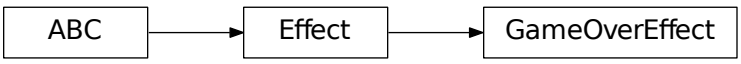
- Play `collect-point.wav`
- Animate Cozmo with `MemoryMatchPlayerWinHand` or `MemoryMatchCozmoWinHand` depending upon `is_player`.
- Flash the cubes green.

Parameters

- **is_sequence_long** – Whether the sequence the player matched was long.
- **is_player** – Whether the player or Cozmo played the correct sequence.

Returns None

song_match.effect.effects.game_over



class song_match.effect.effects.game_over.**GameOverEffect**(*song_robot: song_match.song_robot.SongRobot*)

Bases: *song_match.effect.effect.Effect*

play(*winners: List[song_match.player.Player], did_cozmo_win: bool = True*) → cozmo.anim.AnimationTrigger

Play the game over effect.

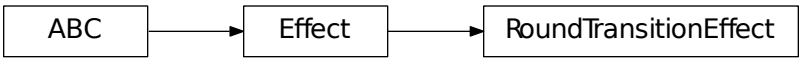
- Play `collect-point.wav`
- Animate Cozmo with `MajorFail` or `DanceMambo` depending upon `is_cozmo`.
- Flash the victory sequence.

Parameters

- **winners** – A list of the players that won the game.
- **did_cozmo_win** – Whether or not Cozmo shared the glory of victory.

Returns None

song_match.effect.effects.round_transition



class song_match.effect.effects.round_transition.**RoundTransitionEffect**(*song_robot: song_match.song_robot.Song*

Bases: *song_match.effect.effect.Effect*

Played when transitioning between rounds of the game.

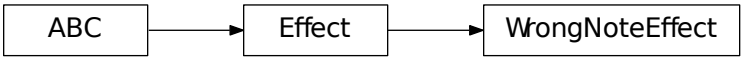
play() → None

Play the round transition effect.

- Play `level-complete.wav`.
- Start the light chaser effect on each cube.

Returns None

song_match.effect.effects.wrong_note



class song_match.effect.effects.wrong_note.**WrongNoteEffect**(*song_robot: song_match.song_robot.SongRobot*)

Bases: *song_match.effect.effect.Effect*

Played when either a player or Cozmo plays the wrong note.

play(*cube_id: int*, *is_player: bool = True*) → None

Play the wrong note effect.

- Play `wrong-buzzer.wav`
- Animate Cozmo with `MemoryMatchPlayerLoseHand` or `MemoryMatchCozmoLoseHand` depending upon `is_player`.
- Flash the incorrect cube red.

Parameters

- **cube_id** – `cube_id`
- **is_player** – Whether the player or Cozmo played the wrong note.

Returns None

song_match.effect.effect

class song_match.effect.effect.**Effect**(*song_robot: song_match.song_robot.SongRobot*)

Bases: `abc.ABC`

Abstract base class for game effects.

song_match.effect.factory

class song_match.effect.factory.**EffectFactory**(*song_robot: song_match.song_robot.SongRobot*)

Bases: `object`

Factory for creating *Effect* instances.

create(*effect_type: str*)

Factory method for creating effects.

Usage: `create('WrongNote')`

Parameters **effect_type** – Upper camel case class name of the effect.

Returns *Effect*

1.7.3 song_match.exceptions

Package containing custom exceptions.

song_match.exceptions.exceptions

exception song_match.exceptions.exceptions.**InvalidEffectType**(*effect_type*)
Bases: ValueError

Raise if an invalid effect type occurs.

exception song_match.exceptions.exceptions.**InvalidGameEffectSound**(*game_effect_sound*)
Bases: ValueError

Raise if an invalid game effect sound occurs.

exception song_match.exceptions.exceptions.**InvalidNote**(*note*)
Bases: ValueError

Raise if an invalid note occurs.

exception song_match.exceptions.exceptions.**MixerNotInitialized**
Bases: ValueError

Raise if constructing a *Note* instance before initializing the mixer.

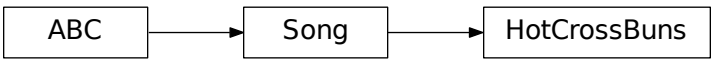
1.7.4 song_match.song

- *Note* - Represents a musical note.
- *Song* - Represents a song.

song_match.song.songs

Package containing *Song* subclasses.

song_match.song.songs.hot_cross_buns

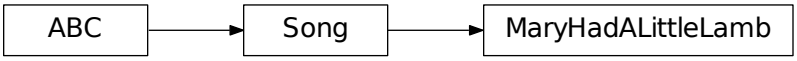


Module containing *HotCrossBuns*.

class song_match.song.songs.hot_cross_buns.**HotCrossBuns**
Bases: *song_match.song.song.Song*

Hot Cross Buns

song_match.song.songs.mary_had_a_little_lamb

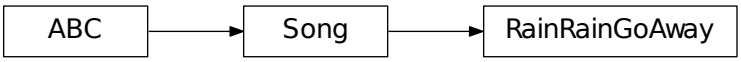


Module containing *MaryHadALittleLamb*.

class song_match.song.songs.mary_had_a_little_lamb.**MaryHadALittleLamb**
Bases: *song_match.song.song.Song*

Mary Had a Little Lamb

song_match.song.songs.rain_rain_go_away



Module containing *RainRainGoAway*.

class song_match.song.songs.rain_rain_go_away.**RainRainGoAway**
Bases: *song_match.song.song.Song*

Rain Rain Go Away

song_match.song.note

Module containing *Note*.

song_match.song.note.**EIGHTH_NOTE = 0.2**
Time for eighth note.

song_match.song.note.**HALF_NOTE = 0.8**
Time for half note.

class song_match.song.note.**Note**(*note: str*, *duration: int = 0.4*)
Bases: object

Represents a musical note.

play() → None
Play the note.

Returns None

song_match.song.note.**QUARTER_NOTE = 0.4**
Time for quarter note.

song_match.song.note.**WHOLE_NOTE = 1.6**
Time for whole note.

song_match.song.song

Module containing *Song*.

class song_match.song.song.**Song**
Bases: abc.ABC

Abstract base class for songs.

Currently only supports songs with 3 notes.

Must override **4** abstract properties:

1. _notes - A list of 3 *Note* instances in ascending order by pitch.
2. _sequence - A sequence of *Note* instances that make up the song.
3. _cube_lights - A list of 3 Light instances.
4. _difficulty_markers - A list of indices where the song ramps up in difficulty.

get_cube_id(*note: song_match.song.note.Note*) → int
Get the Cube ID for a corresponding note.

Parameters **note** – The *Note* of the song.

Returns cube_id

get_cube_light(*cube_id: int*) → cozmo.lights.Light
Get the Light for a corresponding cube.

Parameters **cube_id** – cube_id

Returns Light for the corresponding cube.

get_difficulty_markers() → List[int]
Markers which determine at what position the song ramps up in difficulty.

There are two difficulty markers: 1. Medium 2. and Long

The game starts incrementing by 1 note at a time. Once the game reaches medium, it increments by 2 notes at a time. Once the game reaches long, it increments by 3 notes at a time.

Returns A list of difficulty markers.

get_long_difficulty_marker() → int
Get the long difficulty length marker.

Returns Long difficulty marker.

get_medium_difficulty_marker() → int
Get the medium difficulty length marker.

Returns Medium difficulty marker.

get_note(*cube_id: int*) → song_match.song.note.Note
Get the *Note* for a corresponding cube.

Parameters **cube_id** – cube_id

Returns The *Note* of the cube.

get_sequence() → List[song_match.song.note.Note]
Get the sequence of notes.

Returns A sequence of notes.

get_sequence_slice(*end: int*) → List[song_match.song.note.Note]
Get a slice of the sequence up to and including end.

Parameters **end** – The end position of the sequence.

Returns A sequence of notes up until a certain position.

is_finished(*position: int*) → bool
Returns whether or not the song is finished based upon the position in the sequence.

Parameters **position** – The position in the sequence of notes.

Returns True if the song is finished. False otherwise.

is_not_finished(*position: int*) → bool
Returns whether or not the song is finished based upon the position in the sequence.

Parameters **position** – The position in the sequence of notes.

Returns True if the song is not finished. False otherwise.

is_sequence_long(*sequence_length: int*) → bool
Get whether the length of a sequence is long.

"Long" is defined as being greater than the medium difficulty marker.

Parameters **sequence_length** – The length of a sequence of notes.

Returns Whether the sequence is long

length
Property for accessing the length of the song.

Returns The length of the song.

play_note(*cube_id: int*) → None
Play the note for a corresponding cube.

Parameters **cube_id** – cube_id

Returns None

1.7.5 song_match.sound_effects

Package for custom sound effects.

song_match.sound_effects.sound_effects

song_match.sound_effects.sound_effects.**get_collect_point_sound**() → Sound
Get the collect point sound.

Returns Sound

song_match.sound_effects.sound_effects.**get_level_complete_sound**() → Sound
Get the level complete sound.

Returns Sound

song_match.sound_effects.sound_effects.**get_piano_note_sound_path**(*name: str*) → str

Get the path to a piano note sound file.

Parameters **name** – The name of the note. For example, C4.

Returns The path to a piano note sound.

song_match.sound_effects.sound_effects.**get_wrong_buzzer_sound**() → Sound

Get the wrong buzzer sound.

Returns Sound

song_match.sound_effects.sound_effects.**play_collect_point_sound**() → None

Play collect-point.wav.

Returns None

song_match.sound_effects.sound_effects.**play_level_complete_sound**() → None

Play level-complete.wav.

Returns None

song_match.sound_effects.sound_effects.**play_wrong_buzzer_sound**() → None

Play wrong-buzzer.wav.

Returns None

1.7.6 song_match.config

song_match.config.**ROOT_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/cozmo-song-r**

Root directory of the package to help load .wav files.

song_match.config.**init_mixer**() → None

Initializes pygame's mixer module by calling init().

See https://www.pygame.org/docs/ref/mixer.html#pygame.mixer.init.

IMPORTANT: Must be called before constructing any *Note* objects.

Returns None

1.7.7 song_match.cube_mat

class song_match.cube_mat.**CubeMat**

Bases: object

Class to convert cube IDs to mat positions and vise versa.

Each cube has an ID of 1, 2, or 3 associated with it.

Consider the following cubes with IDs from left to right:

The mat position of each cube from the player's perspective is as follows:

Where the the left-most position is assigned 1, middle position 2, and right-most position 3.

This class is responsible for converting from one to the other.

In this example, the conversion from *cube id* to *mat position* is:

- 2 -> 1
- 1 -> 2
- 3 -> 3

classmethod cube_id_to_position(*cube_id: int*) → int
Maps the `cube_id` to a mat position.

Parameters **cube_id** – `cube_id`

Returns The ordered cube ID.

classmethod get_light_cubes(*song_robot*) → List[cozmo.objects.LightCube]
Convenience method to get a list of light cubes.

Note: This is duplicated in song_match.cube.util due to import issues.

Parameters **song_robot** – *SongRobot*

Returns A list of three `LightCube` instances.

classmethod get_positions() → List[int]
Get a list of mat positions ordered by cube ID.

Returns A list of mat positions ordered by cube ID.

classmethod order_cubes_by_position(*song_robot*) → None
Assign each cube ID to a mat position.

Parameters **song_robot** – *SongRobot*

Returns None

classmethod position_to_cube_id(*cube_id: int*) → int
Maps a mat position to a `cube_id`.

Parameters **cube_id** – The ordered cube ID.

Returns `cube_id`

1.7.8 song_match.game_constants

song_match.game_constants.**MAX_STRIKES = 3**
The maximum number of notes a player can get wrong

song_match.game_constants.**STARTING_POSITION = 3**
The number of notes you start with in the sequence

song_match.game_constants.**TIME_IN_BETWEEN_PLAYERS_AND_COZMO = 1**
Time in seconds to wait in between the players and Cozmo

1.7.9 song_match.option_prompter

class song_match.option_prompter.**OptionPrompter**(*song_robot: song_match.song_robot.SongRobot*)

Bases: object

A class to help the user select an option from three different choices.

get_option(*prompt: str, options: List[str]*) → int

Prompts the user to select from three different options by tapping a cube.

1. Cozmo will prompt the user with `prompt`.
2. Cozmo will point to each cube saying the corresponding `option`.
3. The light chaser effect will start signaling the game is awaiting user input.
4. Upon successful tap `collect-point.wav` is played and the cube flashes green.

Parameters

- **prompt** – The prompt for Cozmo to say.
- **options** – A list of options associated with each cube.

Returns `cube_id` of the tapped cube.

1.7.10 song_match.player

class song_match.player.**Player**(*player_id: int*)

Bases: object

Represents a human player.

did_win

Property for accessing whether the player won the game.

Returns Whether the player won the game.

1.7.11 song_match.song_match

Module containing *SongMatch*.

class song_match.song_match.**SongMatch**(*song: song_match.song.song.Song = None, num_players: int = None*)

Bases: object

Main game class.

play(*robot: cozmo.robot.Robot*) → None

Play the Song Match game.

Pass this function into `cozmo.run_program()`.

Parameters **robot** (Robot) – Cozmo Robot instance.

Returns None

1.7.12 song_match.song_robot

Module containing *SongRobot*.

class song_match.song_robot.**SongRobot**(*robot: cozmo.robot.Robot*, *song: song_match.song.song.Song*)

Bases: object

Wrapper class for Cozmo Robot instance.

did_win
Property for accessing whether Cozmo won the game.

Returns Whether Cozmo won the game.

play_anim(*animation_name: str*, ***kwargs*) → cozmo.anim.Animation
Wrapper method for play_anim().

Parameters

- **animation_name** – The name of the animation.
- **kwargs** – See play_anim().

Returns Animation

play_anim_trigger(*animation_trigger*, ***kwargs*) → cozmo.anim.AnimationTrigger
Wrapper method for play_anim_trigger().

Parameters

- **animation_trigger** – The animation trigger.
- **kwargs** – See play_anim_trigger().

Returns AnimationTrigger

play_note(*note: song_match.song.note.Note*) → None
Make Cozmo play a note.

Parameters **note** – The *Note* to play.

Returns None

play_note_with_error(*note: song_match.song.note.Note*, *sequence_length: int = 1*) → Tuple[bool, song_match.song.note.Note]
Make Cozmo play a *Note* with a chance for error.

Parameters

- **note** – The *Note* to play.
- **sequence_length** – The length of the sequence to play.

Returns Whether Cozmo played the correct note, and the *Note* he played.

Return type Tuple[bool, *Note*]

play_notes(*notes: List[song_match.song.note.Note]*, *with_error=False*) → Tuple[bool, Union[NoneType, song_match.song.note.Note]]
Make Cozmo play a series of notes.

Parameters

- **notes** – The series of notes to play.
- **with_error** – Whether to play the series of notes with a chance for error.

Returns Whether cozmo played the correct notes and the incorrect note he played if any.

Return type Tuple[bool, Union[None, *Note*]]

robot
Property for accessing Robot.

say_text(*text: str*) → cozmo.robot.SayText
Wrapper method for say_text().

Returns SayText

song
Property for accessing *Song*.

tap_cube(*cube_id*) → cozmo.anim.Animation
Make Cozmo tap a cube.

Parameters **cube_id** – cube_id

Returns Animation

turn_back_to_center(*in_parallel=False*) → None
Turn Cozmo back to the center.

Parameters **in_parallel** – Whether to do the action in parallel or wait until it's completed.

Returns None

turn_to_cube(*cube_id: int*) → None
Make Cozmo turn in place until the specified cube is visible.

Parameters **cube_id** – cube_id to turn to.

Returns None

world
Property for accessing world.

song_match.song_robot.**random**() → x in the interval [0, 1).

1.8 Dependencies

This page aims to explain Song Match's dependencies and why we need them.

All project dependencies are listed in requirements.txt in the repo.

Note: Transitive dependencies (dependencies of dependencies) are also listed in this file.

This document will go over only primary dependencies.

1.8.1 Cozmo

The first dependency is the Cozmo SDK.

You can find more information on their documentation.

Package names:

- cozmo

1.8.2 Audio

PyGame is used for playing audio.

We initialize PyGame's mixer module in *init_mixer()* by calling pygame.mixer.init().

Two classes use the pygame.mixer.Sound object.

1. *Note*
2. *Effect*

Package names:

- pygame

1.8.3 Unit Tests and Coverage Reports

For running unit tests and generating coverage reports we use pytest, pytest-cov, coverage and coveralls.

Package names:

- pytest
- pytest-cov
- coverage
- coveralls

1.8.4 Documentation

We use Sphinx to generate our documentation along with a number of other packages.

Package names:

- Sphinx
- sphinx-rtd-theme - Read the Docs Sphinx Theme
- better-apidoc - A version of sphinx-apidoc with support for templating

We also use Graphviz to generate inheritance diagrams. You can download Graphviz here.

The rest of the packages found in requirements.txt are transitive dependencies.

1.9 Graphic Assets

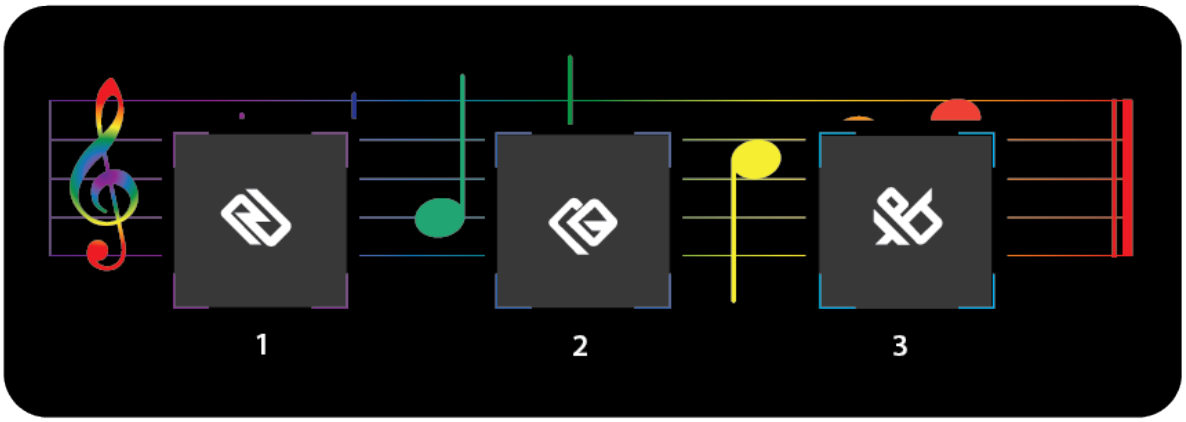
You can download various graphic assets used on this website and in our User's guide as a zip file here:

Download Graphic Assets

1.9.1 Cube Mat

We also made a custom mat to place Cozmo's cubes on.

Download Cube Mat



1.10 Troubleshooting

1.10.1 Problems with Cozmo

If the game does not start after running the correct command:

- Cozmo will not start the game if he can not see the cubes. Most of the time this is because he is not back far enough to see all the cubes, or his arm bar is blocking his view. To fix this, make the appropriate adjustments and the game should start. More information for this can be found in the user guide "Basic Set-up Instructions" on page 5.

`Download User's Guide`

1.10.2 Problems with Cubes

If you tried #1 and the game is still not starting:

- If you tried the above and the game still is not starting, it is possible that the cubes ran out of juice. Try replacing the batteries. From Anki's website, they claim the batteries in the cubes will last about 40 hours. The cubes take a N, E90, LR1 1.5Volt battery. For more information, see How to replace a Cube battery.

1.10.3 Problems with the Application

If Cozmo or the game becomes unresponsive in the middle of playing:

- Close the application and re-launch the game using the python command `python3 main.py`.

1.11 Moving Forward

1.11.1 Cozmo Limitations

During the course of development on this project, we have noticed a few limitations that can affect future additions.

- **Three cube limitation:** We noticed that Cozmo will only recognize the first distinct three cubes placed in front of him. This means he will not recognize additional cubes. So you can't use 5 cubes to open a wider range of songs that could be played.
- **Lighting:** We also noticed, if the lighting is too low, Cozmo's camera can not see the cubes properly. This could cause issues during gameplay. Also, if the lighting is too bright, the players can not see the color of the cubes.

1.11.2 Some Ideas for Moving Forward with the Project

We had many great ideas for this project, unfortunately, the semester is only so long and we could not fit all of our ideas in this package. We coded this project with the thought of making it easy to make changes or expanding in the future.

We originally were going to name the project "Cube Jam" with the thought that we would have multiple games to play based on the sense of hearing.

Other game ideas included:

- An Ear Training game: This would be a game where a note would be played, and the player would have to guess what the note was by tapping the appropriate cube. This game mode could also train relative pitches.
- A duet mode: In duet mode, the player would play along WITH Cozmo, not so much against him. A game based on collaboration and not so much competition.

Other ideas on how to expand could include:

- Add a cube pagination menu for selecting options
- Add additional difficulty levels
- Add more instruments
- Add additional songs
- Add a baton prop that Cozmo could use to point to the cubes

1.12 The Cozmonauts

This project is brought to you by The Cozmonauts.

A team of four undergraduate students at the University of Missouri - St. Louis.

- Kevin Blank
- Ricky Wilson
- Briton Powe
- G Roques

1.13 Documentation

We automatically generate our documentation with a tool called Sphinx.

A Sphinx extension, called sphinx-apidoc, automatically generates Sphinx sources as reStructuredText or `.rst` files from the `song_match` package.

For a primer on reStructuredText see Sphinx's reStructuredText Primer.

Sphinx creates the HTML and other necessary files in `docs/_build`.

We host our documentation using a free service called Read the Docs.

1.13.1 How to Update the Docs

1. Make your desired changes.
2. Run `make html` from `./docs`.
3. Verify your changes by opening `docs/_build/html/index.html` in your favorite web browser.
4. Commit, and push your changes. Read the Docs will update the documentation site upon pushing.

1.14 Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

T

W

Y