Some title for a multimodal chess interface

Giuseppina Iannotti, and Davide Marincione

***Abstract*—This is the abstract.**

I. Introduction

This intro has to change!

In this report, we present the design, implementation, and testing, of a multimodal interface for playing chess. The interface allows the user to play with either mouse movements, hand gestures or voice commands. The user can switch between the different modalities at any time during the game. It is implemented in `python` and uses the `pygame` library for the audio-visual inteface, `opencv` and `mediapipe` for hand gesture recognition, and the `dragonfly` library for voice commands. The interface has been tested with a group of 12 users, and the results show that the multimodal interface is more engaging and fun to use than a traditional mouse-based interface.

II. Coding, Graphics & Audio

Libraries: A chess program is quite complex to make, and since writing one would've been a project in itself, we have decided to use the `python-chess` library. This choice enabled us to abstract away the complexity of the game, and to focus on making the interface and the interaction. `python-chess` is used all over the project's code, to make checks and queries to display the correct information, but also to interact with the chess engine and to make moves.

For the graphics, audio and event handling, we used the `pygame` library, which is a set of `python` modules designed for writing small video games. It is simple yet very powerful, as it allows to draw shapes, images and text on the screen, to play sounds, and to handle user input. Admittedly, it can be quite slow for medium to large-scale projects, as its main drawback is the graphics rendering, which is done via old-fashioned blitting. But, for a project such as ours, it was more than enough.

Programming paradigm: For all its usefullness, `pygame` only provides basic functionalities and none of the data structures and systems used in writing videogames. Things that, to a certain extent, we needed for our project. So we decided to go for an OOP approach, defining ever more refined objects, building on top of more abstract ones. This tactic proved to be extremely successful, as in the later parts of the project there were a couple of instances in which we needed a new class, and we were able to define it without much hassle.

Basic structure: Inspired by many game engines (Unity, Godot, etc.), our first class was the `Object` class, which is the base class for all the elements in our program. It is defined to be as generic as possible: an object in 2d space which, being in a parent-child hierarchy, can have its absolute position changed by the parent, while maintaining its relative position to it.

III. Hand gestures

IV. Voice commands

V. Experiments and Results

VI. Conclusions

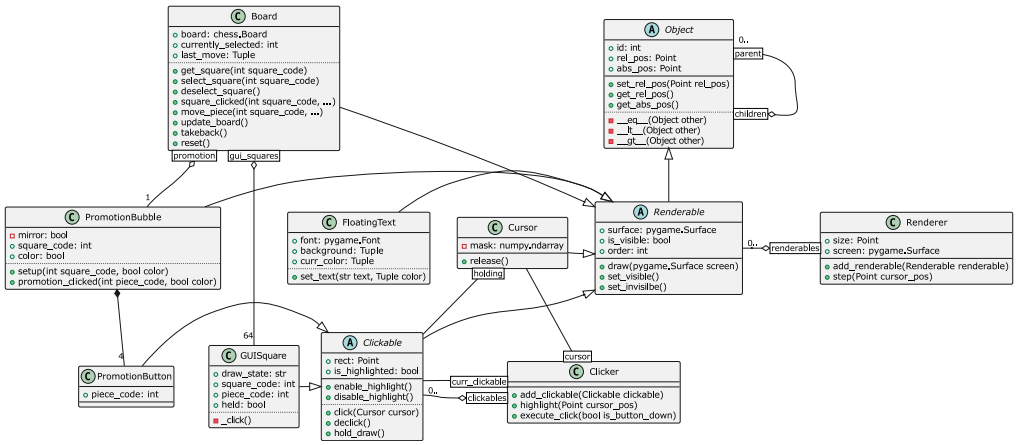

Fig. 1. UML description of the basic structure of the program.