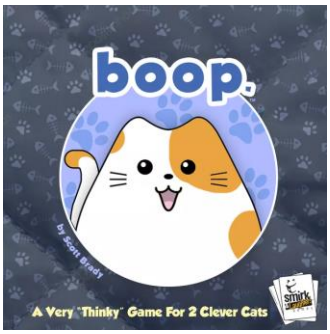
CS 2401 – Spring 2023

Project 6 – A Game

Boop



The final project for CS2401 is a single large project to be submitted in three stages and counts as three projects. For this project we will be implementing a game of Boop, with the final product being a game that can play an intelligent game of Boop against a human opponent.

This game **must** be derived from the files I have given you (a game class written by the author of a textbook we used to use for this course). The files are *game.h* and *game.cc*. I have also included a file called *colors.h* which was created by a former student who has given us permission to use it. It allows you to adjust the colors of the terminal screen during a text-based console or ssh session. I have altered the *play* function in the game class, by commenting out some of the code, so that it will work for the first phase of the project. By the end we will return to the original game class with only a couple of little alterations.

The game class creates a map for us in how the project is developed, and at the end we will find that most of the "AI" has already been written for us in this parent class. If you look at *game.h*, you will see that there are virtual functions that **must be overridden** and some that **may optionally be overridden**. Eventually you will write a child version for all the mandatory overrides, but you probably do not need to override any of the optional ones.

The rules of this game are fully outlined in the Boop_Rules.pdf document included with the assignment. Basically, the game consists of two players that each start with 8 "kitten" pieces. You can upgrade "kitten" pieces to "cat" pieces with the goal of the game being to get three "cat" pieces in a row (horizontal, vertical, or diagonal). The catch is that whenever a piece is added to the board, other pieces may need to move. When placing a kitten any adjacent kitten pieces get "booped" one space away from the newly placed piece if there is an empty spot in that direction. When placing a "cat" piece, all cats and kittens get booped.

The first stage is the **design stage**. In this part you decide how you will represent the pieces and how you will display the board. Good grades are given for the quality of the design, the attractiveness of the board, and the ease of the user interface. This first stage should be derived from the game class. You will create a child class for Boop, which will have a way of storing the board. The board should be a 6x6 two-dimensional array of spaces, pieces, or pointers to spaces or pieces, where the spaces are another class which you have written (I would suggest using a static const for the size of the board so that you can use it later when checking boundaries). This board becomes the principle private member of the Boop class. The spaces class should be able to store all the attributes that a space (or you can call it a piece) might have – emptiness, black, white, as well as mutators and accessor functions to transform a piece/space from one state to another. You will also need counters for how many cat and kitten pieces each player has available to them.

For your pieces, you can stick with the traditional cats and kittens (though you must design your own pieces, don't use the ones that I did). You are also free to be creative with your pieces as long as the two piece types are distinct enough that the players can easily tell them apart.

You should then implement your design to the stage where I can see the board displayed and be allowed to make one initial move. The first step in doing this is to write your space/piece class, which will **not** be derived from anything, but can change states. Then, when you write your Boop class, which will be derived from the game class, the best first step, after declaring your board, **is to create stubs for all the author's purely virtual functions.** (A stub is a function with an empty implementation, which exists merely to validate a call, or to allow the program to compile.) For this stage you should implement *display_status* and a very simplified *make_move,* and *is_legal* which should make sure the move string has the correct format and that it would be legal to put a piece on the space requested. Notice that the author expects the move to be entered and passed around as a string, and you need to stay with this as it is an essential part of the design. Think of the string as merely a container for characters. (Remember you will have inherited the *get_user_move* function – just go ahead and use it.)

Blackboard submission of this 50-point stage (6a) is due at 11:59 p.m. on Friday April 7th.

In the second stage you implement a fully functional two-player game which allows two humans to play the game against each other. (You will have one of the humans "be" the computer in the terms of the author's game class.) **All rules should now be enforced.** This stage will involve much more extensive *is_legal* and *make_move* functions, since now those functions must embody all the rules of the game. This is the hardest stage of this project so you will want to start on this as soon as possible.

Blackboard submission of this 50-point stage (6b) is due at 11:59 p.m. on Wednesday April 26th.

In the final stage the computer will play an "intelligent" game against a human opponent. Again, all rules should be enforced, and the computer should not cheat. This is frequently called the "AI" stage of the game, and you will find that most of the AI work is done by the author's game class. You may find during this stage that the computer makes moves that should not be allowed, in which case you will need to fix your *is_lega*l and possibly *make_move* functions.

Electronic submission of this 50-point stage (6c) is due on Friday May 5th at 1:00 p.m.