

Checkers Game Data Model Concept

Fan Gao, fgvf8@mail.missouri.edu

Description: create a data model concept for the game of Checkers (Draughts).

The Model is the most important part in the MVC framework, and for this Checkers Game, we need to give a clear demonstration on the design of data model.

First, we talk about the rules of Checkers Game.

Checkers is played on a standard 64 square board. Only the 32 dark colored squares are used in play. Each player begins the game with 12 pieces, or checkers, placed in the three rows closest to him or her. The object of the game is to capture all of your opponent's checkers or position your pieces so that your opponent has no available moves.

Movement

Basic movement is to move a checker one space diagonally forward. You cannot move a checker backwards until it becomes a King, as described below. If a jump is available, you must take the jump, as described in the next question and answer.

Jumping

If one of your opponent's checkers is on a forward diagonal next to one of your checkers, and the next space beyond the opponent's checker is empty, then your checker must jump the opponent's checker and land in the space beyond. Your opponent's checker is captured and removed from the board.

After making one jump, your checker might have another jump available from its new position. Your checker must take that jump too. It must continue to jump until there are no more jumps available. Both men and kings are allowed to make multiple jumps.

If, at the start of a turn, more than one of your checkers has a jump available, then you may decide which one you will move. But once you have chosen one, it must take all the jumps that it can.

If a jump is available for one of your pieces, you must make that jump. If more jumps are available with that same piece, you must continue to jump with it until it can jump no more. To make the second and third jump with a piece, you do not need to click that piece again. Just click the next space to which it will jump.

If more than one of your pieces has a jump available at the start of your turn, you can choose which piece you will move. But then you must make all the jumps available for that piece.

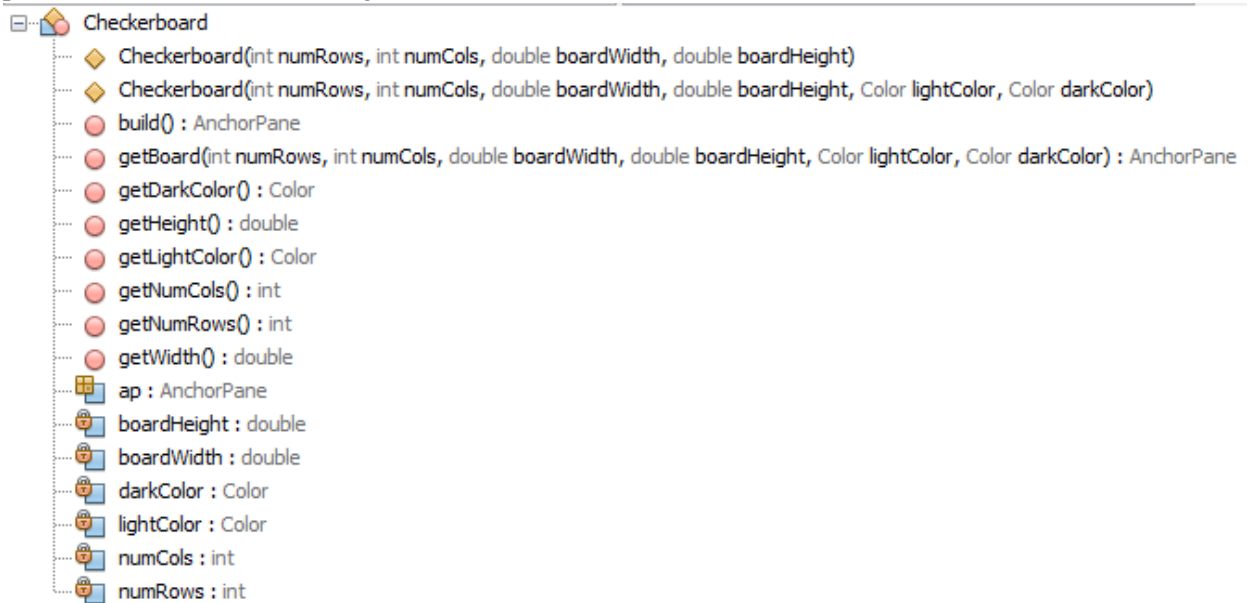
Crowning

When one of your checkers reaches the opposite side of the board, it is crowned and becomes a King. Your turn ends there. A King can move backward as well as forward along the diagonals. It can only move a distance of one space. A King can also jump backward and forward. It must jump when possible, and it must take all jumps that are available to it. In each jump, the King can only jump over one opposing piece at a time, and it must land in the space just beyond the captured piece. The King cannot move multiple spaces before or after jumping a piece.

Data Model Design

1. Checkerboard

The first thing is building a checker board object and visualize it in the View part, just like the previous checkerboard challenge.

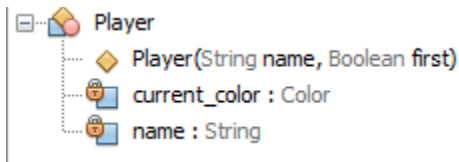


After this, we will have an 8*8 checkerboard, actually this part will be showed as a User Interface in the View part. In the latter design, a 2D array will cover the whole checkerboard, using the index to locate a piece and it's movements.

2. Player

Basically, player part need a name and a priority of moving the piece.

So here the function will receive a Boolean value to check if this instantiated player would move first.



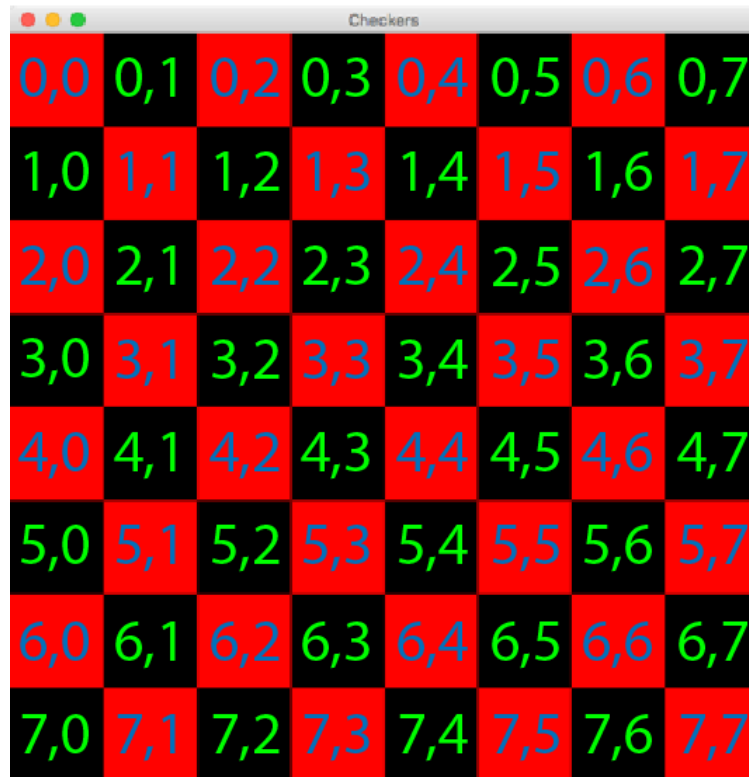
The name parameter will also be received and the color will be set after the judgement of Boolean value.

```
public Player(String name, Boolean first){
    this.name=name;
    if(first){
        this.current_color=Color.BLACK;
    }else{
        this.current_color=Color.RED;
    }
}
```

3. Movement and Jump

Here we use the 2D array to present the location of every piece, and every move or jump will be implemented in the way of index transition.

Dark Player



0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7

Light Player

We use the index calculation to predict all the possible next step, if the requirement is met, the move or jump would be valid, otherwise warning of invalidate operations would be showed in the screen.

The logical idea of move and jump:

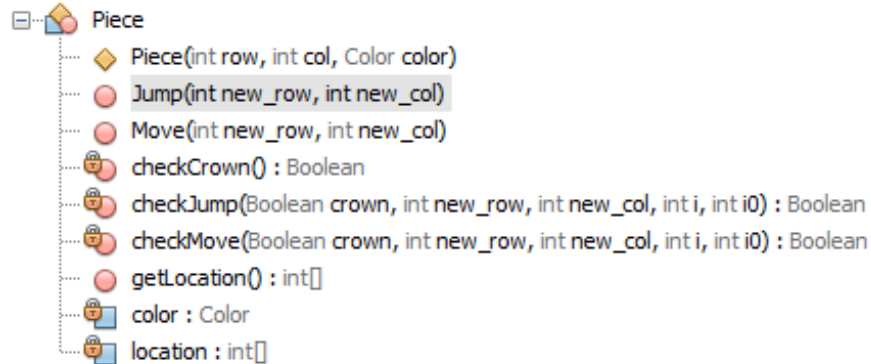
```
public void Move(int new_row, int new_col){
    Boolean crown=checkCrown();
    Boolean legal=checkMove(crown,new_row,new_col,location[0],location[1]);
    if(legal){
        location[0]=new_row;
        location[1]=new_col;
    }
}

public void Jump(int new_row, int new_col){
    Boolean crown=checkCrown();
    Boolean legal=checkJump(crown,new_row,new_col,location[0],location[1]);
    if(legal){
        location[0]=new_row;
        location[1]=new_col;
    }
}
```

```

    }
}

```



In this way, every piece will have an exact next behavior, if nothing valid on the next step, the player will be judged as loser.

Check the crown state:

```

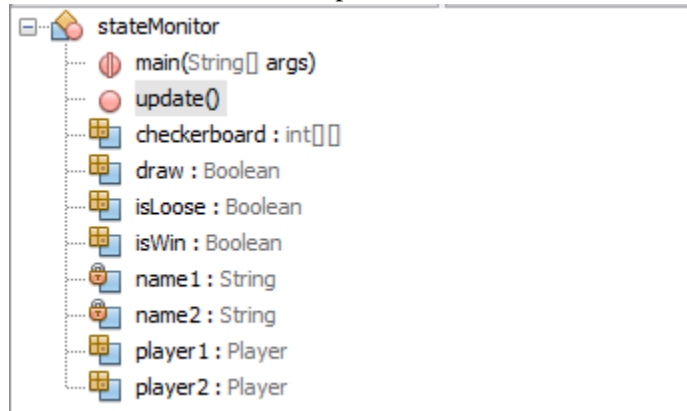
private Boolean checkCrown() {
    if(location[0]==0&&location[1]%2==0){
        return true;
    }else{
        return false;
    }
}

```

After the crown state is checked, the next legal step will be easily implemented.

4. Game State Listener

The listener's job is to monitor the piece's movement. If the piece touches down, it becomes a crown; if the piece cannot move, it loose; if no pieces left, it loose, etc.



Here two players will be instantiated, at least one will be the user, and another one is AI or user. And the update function will check the state of pieces to determine the current state of game. About the draw condition, after the opponent's conformation, the current location will be rolled back to the previous location, and all the relevant states need rolling back.

So, after the design of above four parts, the data model of Checkers Game is generated. Although it is just a rough idea, some design patterns do have a potentiality to be implemented. The data model is the reference to present the data, and all the functionalities will count on the change of relevant data structures, like the state machine. The controller part will monitor the data and assign a proper response event as feedback, and the view part will show the result intuitively, which is how the MVC framework collaborate with each other.