

**NATIONAL COLLEGIATE DODGEBALL ASSOCIATION**



**N-DROPP**

**Detailed Design Document (D3)**

**Kyle Peltier**

Lead Programmer/Designer

[peltikyl12@gmail.com](mailto:peltikyl12@gmail.com)

**Zigmas Maloni**

NCDA League President

[ncdadodgeball@gmail.com](mailto:ncdadodgeball@gmail.com)

**Contributors:**

Mario Galeno – UX Design

Matthew Lukasiewicz – UX Design

Matthew Williams – UX Design

**Contents:**

[N-DROPP Source Design 3](#_Toc377654820)

[Model-View-Controller (MVC) 3](#_Toc377654821)

[Documentation 3](#_Toc377654822)

[Files 3](#_Toc377654823)

[Classes 3](#_Toc377654824)

[Methods 4](#_Toc377654825)

[Notes/Comments 4](#_Toc377654826)

[ShotClock FSM 5](#_Toc377654827)

[GameClock FSM 7](#_Toc377654828)

[RANDOM JIBBERISH (NOTES,REMINDERS,ETC.) 8](#_Toc377654829)

# N-DROPP Source Design

## Model-View-Controller (MVC)

The N-DROPP coding structure will be in MVC form. I’d like each segment of the design to have its own thread, therefore making this a multithreaded program.

The Model thread will be the main thread of the application. This thread will make sure all of the brainiack work is going smoothly.

The View thread will continually check the values of the Model, updating its displays based on those values. This will ensure that the graphics of the application are constant and smooth.

It’s difficult to get the Controller to be its own thread but input from the user will directly modify the Model and updates to graphics will follow.

**Removed MVC design structure. A thread that did not create a View object cannot modify it. Hence, can’t have a “View” thread.**

# Documentation

## Files

All files should begin with the following header:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \_\_\_\_\_ \_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* \_\_\_ | / / \_\_\_ \_\_ \\_\_ \_\_ \\_ \_\_ \\_\_ \_\_ \\_\_ \_\_ \

\* \_\_ |/ /\_\_\_\_\_\_\_\_ / / /\_ /\_/ / / / /\_ /\_/ /\_ /\_/ /

\* \_ /| /\_/\_\_\_\_\_/ /\_/ /\_ \_, \_// /\_/ /\_ \_\_\_\_/\_ \_\_\_\_/

\* /\_/ |\_/ /\_\_\_\_\_/ /\_/ |\_| \\_\_\_\_/ /\_/ /\_/

\*

\* National Collegiate Dodgeball Association (NCDA)

\* NCDA - Dodgeball Referee Officiating Application

\* http://www.ncdadodgeball.com

\* Copyright 2014. All Rights Reserved.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

The ASCII art is made possible by <http://patorjk.com/software/taag/>.

## Classes

Each declared class should contain a description comment in the format of the one below:

/\* ClassName

\* Class description should go here. This should be a brief overview

\* of the entire class. All class names should begin with a capital letter.

\* Notice the details of this comment. The comment is started by “/\*”

\* (single asterisk) followed by the class’ name. The next line should

\* start the description text.

\*/

public class ClassName

{

...

}

## Methods

Each declared method should contain a description of the method, its parameters, and the value it’s returning (if any).

/\*\* int methodName ( arg1, arg2 )

\* @param arg1 : very brief description of argument 1

\* @param arg2 : very brief description of argument 2

\*

\* @return description of the value that will be returned

\*

\* Method description goes here. All methods should begin with a lowercase

\* letter. Method comments should begin with “/\*\*” (double asterisk).

\* If the method lacks parameters or a return type, omit them from the

\* comment as necessary.

\*/

public int methodName(int arg1, String arg2){

...

}

## Notes/Comments

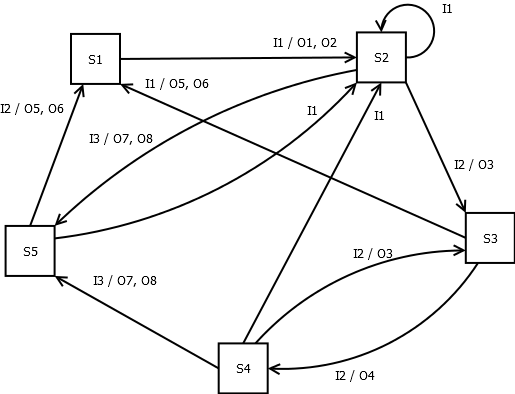
Other source code comments can be denoted either by block comments (/\* \*/) or line comments (//). If you’re using block comments, make sure the comment begins with a single asterisk (/\*) and not two (/\*\*) as denoted by Class comments.

## ShotClock FSM

|  |  |  |
| --- | --- | --- |
| **States** | **Name** | **Description** |
| S1 | Paused Top | Clock is not running and its current time starts at the top (full time) |
| S2 | Rolling Top | Clock starts running from the top (full time) |
| S3 | Paused | Clock is paused |
| S4 | Resumed | Clock resumes counting from a pause |
| S5 | Expired | Clock timer has reached zero |

|  |  |  |
| --- | --- | --- |
| **Input Event** | **Name** | **Description** |
| I1 | B1 - Reset/Start/Restart | (Button 1): Single input button that represents Start/Reset. (Restart when expired) |
| I2 | B2 - Pause/Resume/Reset | (Button 2): Single input button that represents Pause/Resume (Reset when expired) |
| I3 | Expired | Triggered event when clock reaches zero |

|  |  |  |
| --- | --- | --- |
| **Output Event** | **Name** | **Description** |
| O1 | B1🡪”reset” | Change B1’s text to “Reset” |
| O2 | Enable B2 | Allow B2 to be clickable |
| O3 | B2 🡪”resume” | Change B2’s text to “resume” |
| O4 | B2 🡪 “pause” | Change B2’s text to “pause” |
| O5 | Disable B2 | Disable B2 from being clickable |
| O6 | B1 🡪 “start” | Change B1’s text to “start” |
| O7 | B2 🡪 “reset” | Change B2’s text to “reset” |
| O8 | B1 🡪 “restart” | Change B1’s text to “restart” |

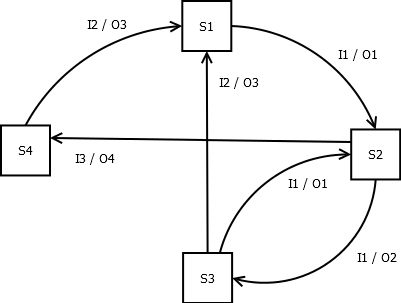


## GameClock FSM

|  |  |  |
| --- | --- | --- |
| **States** | **Name** | **Description** |
| S1 | Paused Top | Clock is not running and its current time starts at the top (full time) |
| S2 | Running | Clock is running |
| S3 | Paused | Clock is paused |
| S5 | Expired | Clock timer has reached zero |

|  |  |  |
| --- | --- | --- |
| **Input Event** | **Name** | **Description** |
| I1 | B1 – Start/Pause/Resume | (Button 1): Single input button that represents Start/Pause/Resume |
| I2 | rolloverTime/Halftime | Event indicating the current time should be rolled to next half |
| I3 | Expired | Triggered event when clock reaches zero |

|  |  |  |
| --- | --- | --- |
| **Output Event** | **Name** | **Description** |
| O1 | B1🡪”pause” | Change B1’s text to “Pause” |
| O6 | B1 🡪 “resume” | Change B1’s text to “Resume” |
| O7 | B1 🡪 “start” | Change B2’s text to “Start” |
| O8 | Disable B1 | B1 is grayed-out and cannot be selected |



# RANDOM JIBBERISH (NOTES,REMINDERS,ETC.)

IMPLEMENT MODEL-VIEW-CONTROLLER!!! The Shot Clock class can remain the model. Have a variable be current time and update it every callback from the running clock thread. Create a view class that handles the viewing of the clock which will run a thread that constantly reads the clock time from the clock object and then updates a view with the time.

A number of respondents agreed that a stopwatch shot-clock would be a great way to ensure that each team's counter is in sync.

In the open-response question, many individuals noted that they get distracted from watching the game when players argue certain calls. This reenforces the statistice that 35% of people think calling a player out causes the most distraction. Having a quick rulebook to settle the issue or a non-arguable whistle blow could help keep referees on task.

A respondent had an admirable note that the projected application would be great for tournaments. The in-game history will help referees to remember current-game statistics better.

A respondent noted that a rulebook would also be good to have for referee officials to become more familiar with the rules

A few individuals noted that they may be more distracted by a mobile application than benefitted by it while officiating a game. This could lead to audio-related options, taking less of the referee's visual attention.

Some respondents agreed that a built-in whistle would be beneficial, especially for use at practices.

A respondent suggested the application include a visual queue of knocked out players to help referees know the order in which players go back on the court when catches are made.

A respondent noted that the application's shot clock should be fast and smooth, making it quick and easy for the referee to reset the clock without taking much attention away from the game.

In addition to a rulebook, suggested features included search functions, filters, and categorization for easier rule look-up.

An answer to a free-response question suggested recording certain game scenarios where discrepencies unveiled. This way, captains and referees could go back and review these situations and make reformed decisions. This is similar to the idea of recording event history of a game, however, this implementation would require more detail (and effort on the referee's part).

Allow Head referee to end the game early (2 minutes left, just end the game and submit the scores). With this, allow a way for the user to recover a game in case the user accidentally ends the game. To recover a game, it must be the same date as the game was started (or less than 6 hrs if rolled over midnight or something ridiculous like that). In order to recover a game, information such as Players remaining for both teams, game clock, and scores will have to be stored.