# The Game of Yatzy

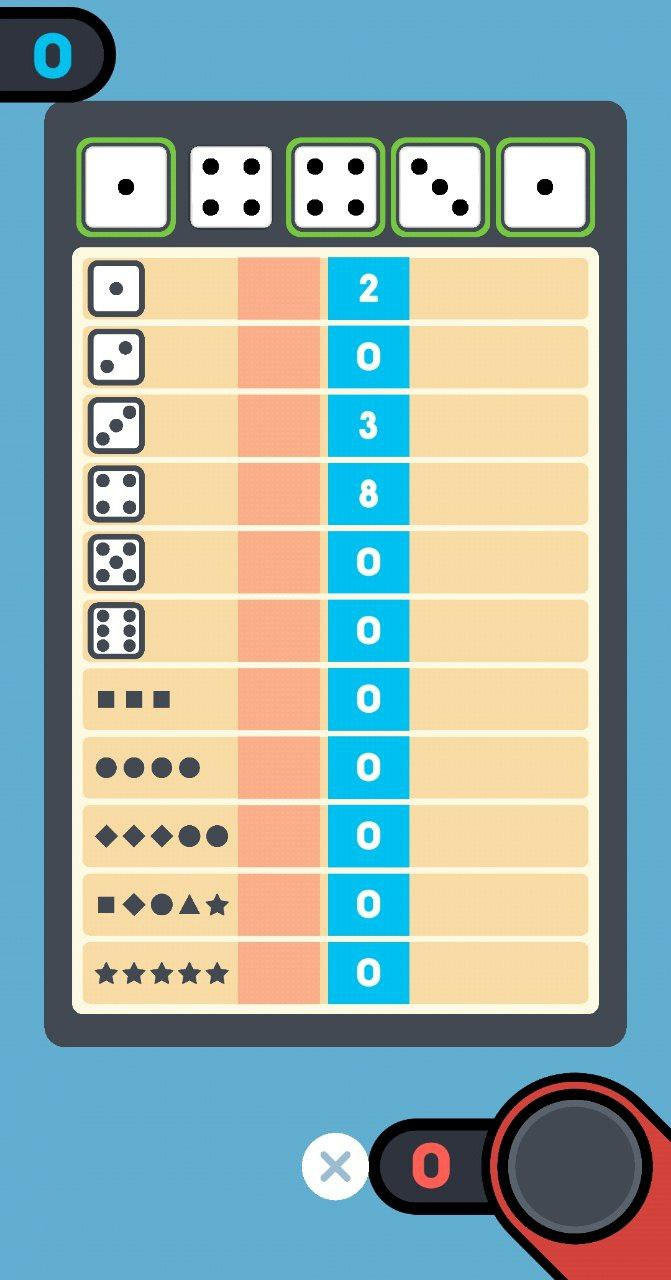
## Vybortseva Anastasia, Mashkin Nikolay, 2025

# Summary

This project implements a game of Yahtzee, which is better known as dice poker. The software is implemented using Cdm-16 and consists of a fully working opponent who should make the most favorable moves based on the dice rolled and the combination patterns occupied. The hardware should be implemented using Logisim and perform the gameplay, including the start of the game (i.e. at the beginning of the game it should be determined who will move first), move transition, scoring, end of the game (when there are no free combination patterns left, the game ends).



# Preliminaries

Above you can see a picture that illustrates an example of the playing field for Yahtzee. This picture is taken from the application “1 2 3 4 Player Games”. And as you can see, this field has 11 patterns for combinations. Next to the playing field, 5 dice are thrown alternately by all players.   
  
According to some version, this game was invented by a married couple from Canada a little earlier than 1956, and called it Yacht Game, because the couple usually played it with their friends on the yacht.

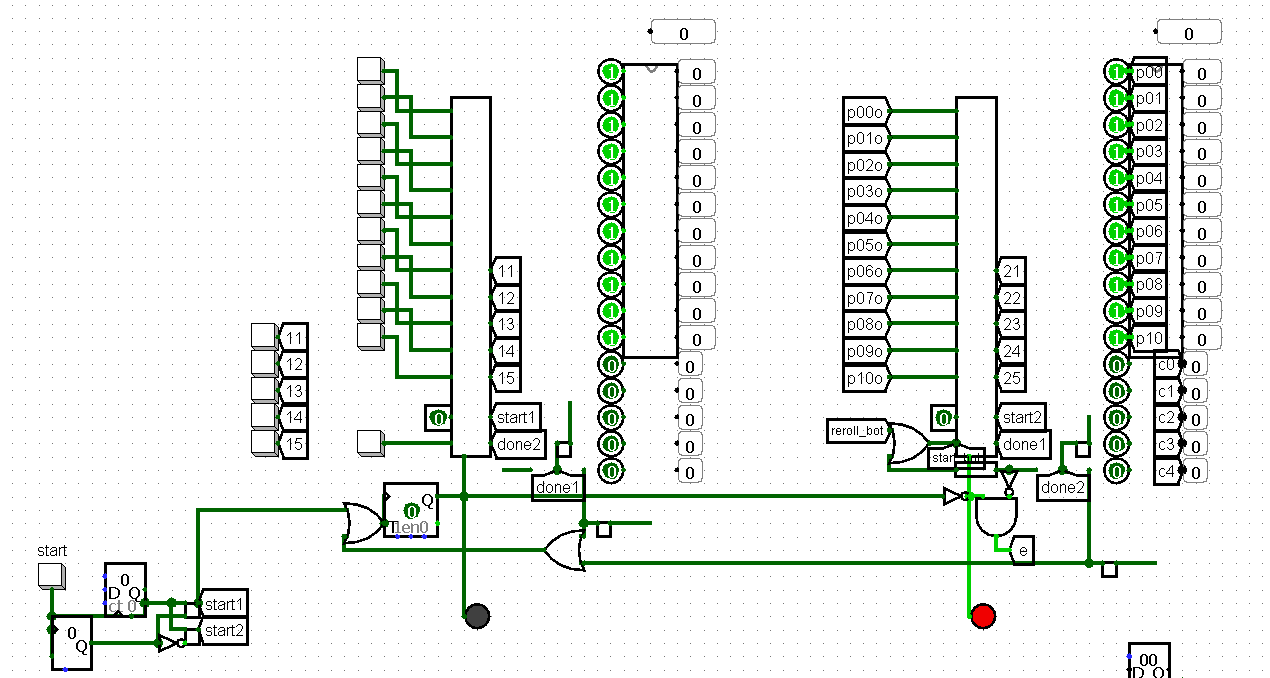
In 1956, they introduced the game to an American entrepreneur, who bought the rights to it and renamed it Yahtzee. However, the game was not initially a success because it seemed too complicated for people, but later the entrepreneur managed to popularize it, and now the game is known in many different variations all over the world.

*Below are the rules for one of the most popular versions of the game:*

1. Two players play the game sitting across from each other at a table. In front of each player there are separate blank slips, where the players will write down the points obtained on each of their moves. The players also have five dice, which they will take turns throwing on the table.
2. The main objective of the game is to score more points than the opponent.
3. The main objective of the turn is to score the most points in the current turn, based on what dice are rolled and what combination patterns are still free.
4. Each turn consists of three dice rolls:
   1. The player rolls the dice on the table.
   2. Based on which dice he rolled, he tries to assemble some combination.
   3. On the current roll, he can fix any number of dice and roll the un-fixed ones two more times, based on the same rules.
   4. After the third die roll, the player must choose any combination from the free ones (each combination can be chosen only once). The turn passes to another player (step 4.1). The action is repeated until all patterns of both players are occupied.

In the end, the player with the highest score at the end of the game wins. The second player is considered the loser. If the number of points is the same, the game is considered to have ended in a draw.

# Game Pad



The schematic you can see above is a logical implementation of the playfield for Yahtzee. There are three blocks that need to be developed for it:

1. Button driver.
2. Dice driver
3. Pattern driver.

*Button driver:*

You have to implement 11 buttons for pattern locking, 5 buttons for dice locking and 1 button for rolling the dice.

*Dice driver:*

Dice display with 5 numbers showing dice values (if 0, it is considered inactive) and 5 lock indicators

*Pattern driver:*

Pattern display with 11 number showing current pattern values and 11 lock indicators

All of them should be connected to the main player element, with the following inputs:

1. Eleven pattern locks, which are triggered via a pulse and upon triggering, “lock” the corresponding pattern, adding the appropriate number of points to the total score.
2. Five dice locks, which are triggered via a pulse and upon triggering, “lock” the corresponding die, so that on the next roll it will not be affected. Also, each roll should reset all the locks dice locks
3. One dice roller, which is triggered via a pulse and upon triggering, rolls all the dice, that are not locked.
4. Start input, indicating that this player is starting the game
5. Done input, indicating that the other player is done and this player can begin making his moves
6. Enable input, indicating, that this player is active

And following outputs:

1. Eleven pattern values and lock indicators, showing the appropriate data
2. Five dice locks and values, showing the appropriate data
3. Done output, to show that this player is done with his moves
4. Roll enabled, to show that this player is able to roll (important for the bot)
5. Swap output, to show that it is time to switch turns

# Possible difficulties

Although Logisim is a great simulator, it is certainly not ideal, so many strange and bizarre things can happen during the process of development of the schematics.

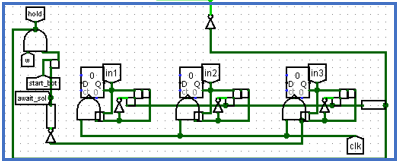
For example, you should watch out for using too many “matryoshka style” schematics, because on less powerful PCs it can cause some wires to turn blue for no reason or just completely ignore some logic in your schematics.

Also, it is important to mention that there are several versions of Cdm-16 processors available on the Internet and they are very different.

First tip is to not use the version, that was implemented in Logisim, because it is extremely slow and your tick rate will hardly be able to climb above 150. Jar versions are much better and faster, as they are implemented in java.

Another advice is to use 0.2.1 version of a processor, because is has a working hold input.

One of the main concerns will be the implementation of data storing amidst the running of the processor. We recommend to use this schematic



Which is essentially 3 timers, that are being triggered when bot has the ability to move and each one of these timers give out a short pulse upon finishing, so that all the data can be stored consecutively.

# Software

Basically, the actual challenge of this project lies in the bot algorithm for identifying the best possible move right now and also identifying, which pattern to lock.

All the work should be done via **play** subroutine, which takes 3 arguments: cubes state, patterns state and end flag.

*Cubes state:*

A 16-bit integer, which has 5 3-bit cube values packed into it. Bits 0-2 represent the value of the first die in the little-endian format, bits 3-5 represent the second die and so on. Bit 15 is left unused.

*Patterns state:*

A 16-bit integer, which has the pattern lock indicators packed into it. Bits 0-10 represent the lock indicators of patterns 1-11

*End flag:*

If set, the bot can no longer reroll and has to decide, which pattern to lock

At the end, the decision should be returned as a 16-bit integer in the following format:

If bit 15 is set to 1, bot has decided to lock the pattern, the index of the pattern is stored in bits 0-3

Else, bot has decided to reroll, bits 0-4 indicate if corresponding cubes should or shouldn’t be locked.

It is suggested, that **main** subroutine should be implemented, with an infinite loop with another loop that processor exits only if an input is given, then it calls the **play** subroutine and then jumps back to the input loop.

*Suggested strategy:*

1. If no more rerolling is possible, lock the best pattern (max points). Else, move on to step 2
2. Check, whether there is an available pattern amongst the “premium” ones (last 5). If there is and there is no better premium pattern available, lock it immediately. Else, move on to the step 3
3. Identify the max number of repetitions among the dice and lock the corresponding dice. If there are more than one dice values with the same max repetition, lock the ones with the highest value and reroll.

Although it may not be perfect, it is good enough and more importantly, it is much faster that trying to properly compute all of the possible combinations and estimate all the probabilities.

But feel free to add some tweaks to make the strategy more accurate.

# Suggested design progression

The hardware and software parts can be easily done separately and connected only towards the end of development, so a great idea would be to implement them in parallel. It is also important to note that it will be much easier to write code in C and then compile it into assembly, as it saves a lot of time. Here is one way to go about it:

*Hardware:*

1. Start with implementing the D6 schematic for properly random dice rolls
2. Move on to creating the roll and cube lock handler.
3. Take care of all the patterns and locking of these patterns
4. Proceed to connect parts from step 2 and step 3, adding logic for rerolling no more than 2 times
5. Finally implement the main schematic, connecting two player panels

*Software:*

1. Start by implementing functions for verifying all the patterns.
2. Move on to function for deciding, what cubes to lock.
3. Proceed with the function for actually making the decision
4. And finish with data unpacking and packing function (play subroutine) to make it convenient for using

*Connection:*

1. Make sure, that your processor is properly attached to the memory
2. Identify the addresses of inputs in memory and set up the system to load the data in
3. Make data packers/unpackers in Logisim
4. Compile the bot.img from your code

And congrats! The project is done!