The Game of TV-Tennis

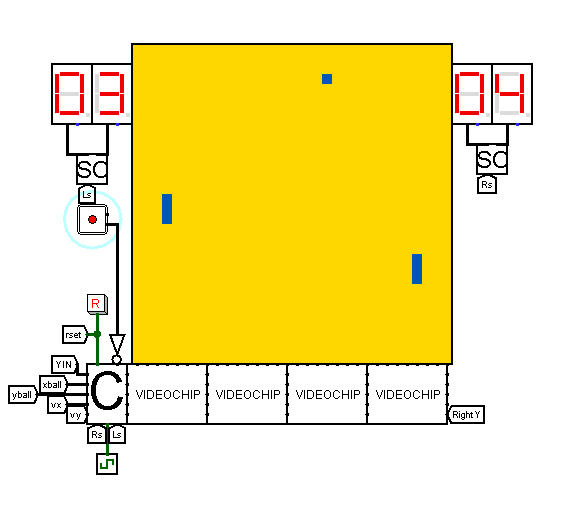
Maxim Chernyshov, Timofey Khrapovitskiy, Artyom Solovyov

2022

Overview

This project was created in 2022 at Novosibirsk State University during the course "Digital Platforms". It is a game of Tv-tennis, which was created using Logisim(hardware) and CDM-8(software). The goal of this project was to create a game against the AI, which would be interesting to play and that it would be possible to beat the AI.

On the screen there are 2 racquets that can move up and down, but not left and right. The ball can move in all directions, the player's goal is to hit it and not let it touch the side of the screen behind his racket. if he fails to hit it, his opponent (in our case the AI) gets a point. The player scores points in the same way - it is necessary that the AI fails to hit the ball. In order to balance the game, the AI racket can only move when the ball is halfway up the player's screen, at other times it always stands still. In our case, the player's racket will be on the left and the AI racket will be on the right. Under these conditions, the player has a chance to win.



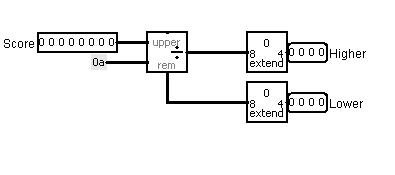
Display

The screen consists of 1024 pixels and has the shape of a square (32x32). Each column receives a 32 bit input pin, one bit per pixel. Each pixel can be lit or unlit. In order to display 2 racquets and a ball, we need the coordinates of the ball and the Y coordinate of both racquets. The racquet consists of 3 pixels going in a row, lit in the same column. But we only need a central one, on either side of which we light the pixels together, getting the racquet.

3.2) In order to display the screen we need a video system. It consists of 32 video chips, which are combined into 4 sections of 8. Each video chip is responsible for its own column numbered from 0 to 31. We number them so that they know which column they are responsible for. This works like this: each chip has an extra input in which their ID is written. We add 1 to the chip ID as we go through the video system. so we give the first chip ID 0b00000 and the chips get the correct numbering. the chip also takes the ball and Y coordinates of the right and left racquet. The special columns in the section are 3 and 4. Column 3 is the column with the left racquet (in section 0) and column 4 is the column with the right racquet (in section 3). The ball can be in any point on the screen, so both the ball and the racket can be in these columns at the same time.

Score panels show scores of players, which they get from the kinematic controller.

Scores scheme split the score into two to turn it into a decimal number on the panel.



Kinematic controller

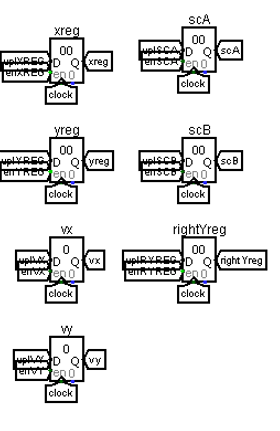
This component of the project updates the position of the ball, blocks the right racquet in the right side of the screen, and sends coordinates of the ball and racquets.

These registers are needed to store scores, coordinates, velocity of the ball and the Y coordinate of the racquet.

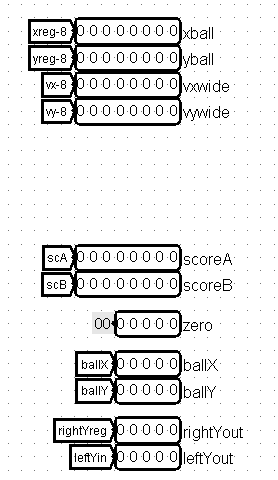
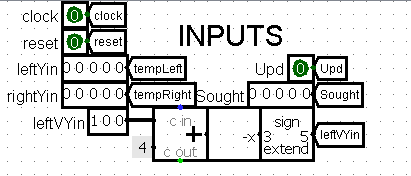
The scores are 8-bit, coordinate of the right racquet is 5-bit, it’s a value from 0 to 31 — the pixel on the screen.

The ball coordinates are 16-bit. The 8 upper bits are the balls coordinate on a field with size of 256. The other 8 bits are needed for better accuracy.

The velocities are 12-bit. Every tick velocities are added to coordinates. To perform so, they are giver extra for zeroes as higher bits. If a velocity is 256, it means that every second the ball goes 1/256 of the field.



In contrast to the displayed screen, the size of the space to move the ball and calculate its coordinates is a square 256x256. So the display will be such that a pixel with coordinates (0;0) will represent any position in a square (0-7, 0-7) on that grid.  Because of the way the screen is displayed, we can easily turn 16-bit ball coordinates for the kinematic controller into 5-bit coordinates for the video system. We take up to 5 significant bits with the highest value, and if they are less than 5, we refill them with zeros. The biggest error is the sum 1+2+4 = 7 in the case of 255. 255-7=248 is still in range of the top right square. The racquet Y coordinate is a 5-bit value, so we don't have to worry about that.



The controller’s input is: clock generator, reset button, previous values of the bats’ Y coordinates, left bat’s velocity. The last are the best coordinate for the right bat from the processor and the flag to save this best coordinate.

For the processor it return 8 most significant bits of ball’s x and y coordinates and bits from 1 to 8 of velocities.

It also outputs the score. For videochips the controller returns bat coordinates, 5 most significant bits of ball coordinates and 0 number for the first video chip.

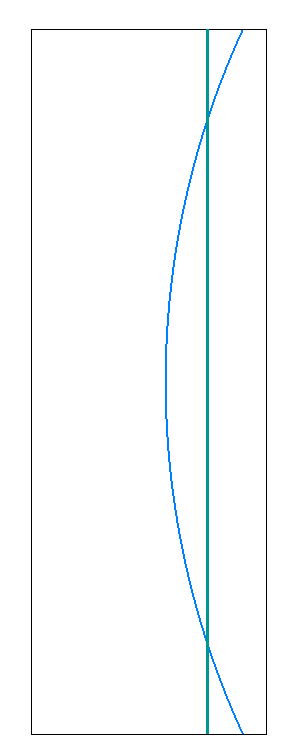
Movement of the ball is iterative. Every tick the ball updates it’s coordinate and, if some conditions are true, velocity. The schemes take values stored in registers and given as inputs. They calculate the values for the next step. Then when clock goes low, the registers are updated.

How do we check if the ball collides with a wall? If vx is lower than 0, when we add it to x coordinate, carry should be 1. If it is zero, then the collision happened. If vx is higher than 0, carry should be 0, otherwise the collision happened. The same way we detect vertical collisions. When we detect any kind of collision, we negate the proper velocity and instead of continuing to move with the same velocity the ball bounces in the opposite direction. The corner collision is just two collisions at the same time.

Each time a ball bounces off a horizontal wall, player opposite of that wall receives a point. When the carry bit is detected, the update of the score counter is made in the second step of a clock cycle. If vx is greater than zero, the first player gets a point and vise versa.

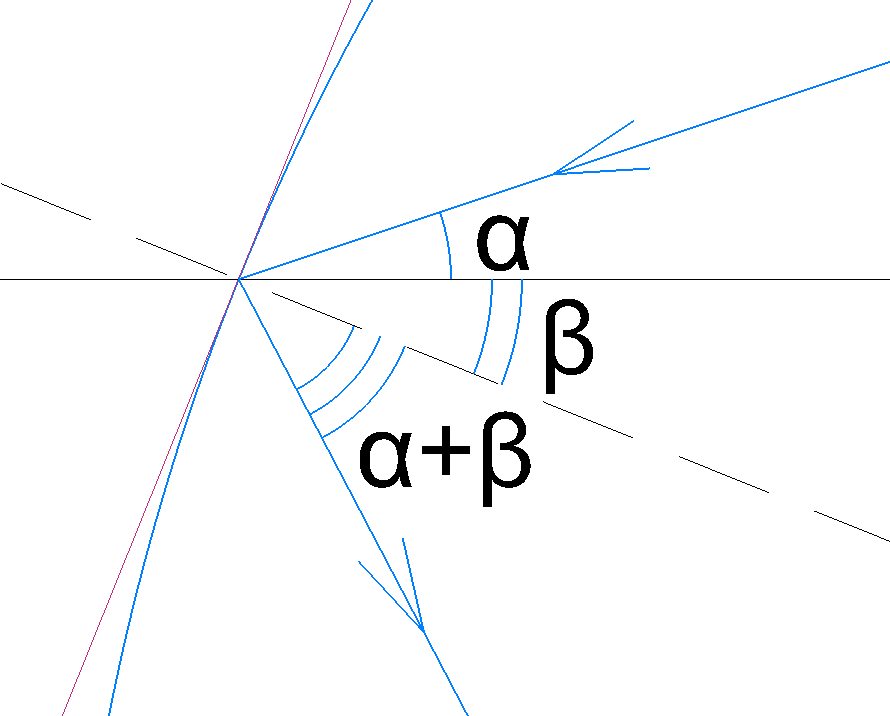
Bouncing the ball of the bat

The bats are circular arcs whose. They are horizontally symmetrical and the centres of their circles are closer to the middle of the field. We need this information for calculating angles of reflection. But first, we need to find the moment when the ball collides a bat. For that purpose, we assume that as a vertical segment. The left bat is drawn on 4th line of pixels, that means it seems to be a rectangle with x coordinate from 24 to 31 out of 256, but the segment’s coordinate is x=30. The right ball’s x is 226 while it’s rectangle is from 224 to 231.



It the ball collides the bat, two conditions are true. First, calculate xball+(256-xbat). If when adding vx to this produces overflow, that means that the ball went through the vertical line whose x is xbat. Second, we calculate the distance from the bat’s bottom y to the ball’s y. If it is less than 8\*3, than the ball is on the same height with the bat. There is a check that the ball and the bat aren’t in opponent edges of the field, else there might be fake hits because of overflow.

The distance between the bat’s bottom and the ball is the place on the arc, where the ball’s trajectory goes through it. Assume that the angle between the trajectory and x+ line is . The angle between the radius of the arc at that point and the x+ axis is . Both the inception angle and the reflection angle are **.** The angle between the trajectory after the reflection and x+ is .

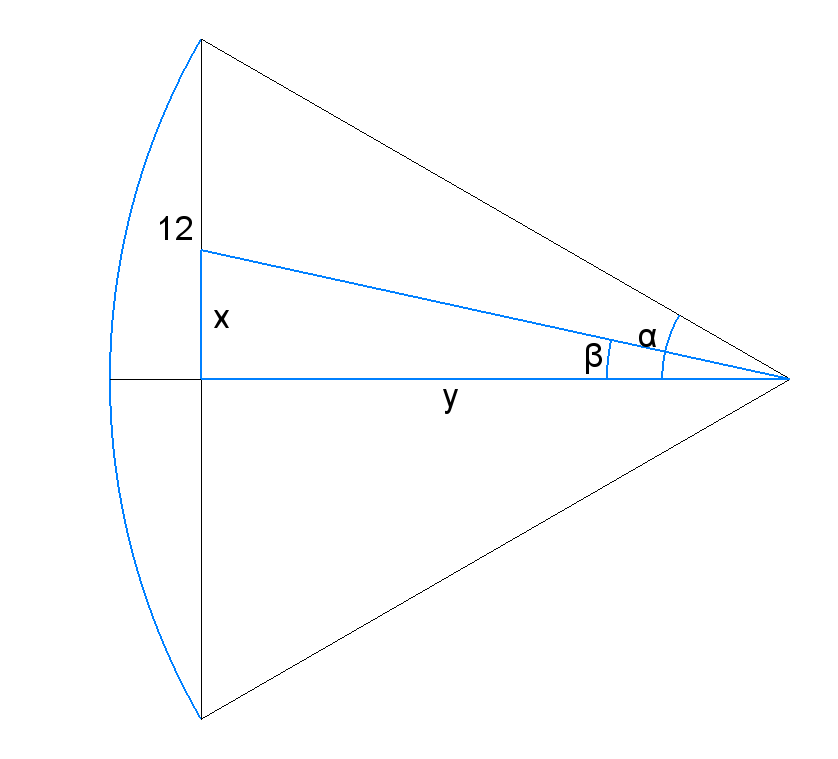


Both before and after the reflection the ball’s speed is the same. Assume it’s v. is the projection of ball’s speed on x coordinate before the reflection, is the projection for y. and are the projections after the hit. We can express these variables through v and trigonometric functions.

Let’s express vx’ and vy’ through other values

We know and . We also know what point ark the ball hit. If and are calculated for every possible points, and can be easily calculated.

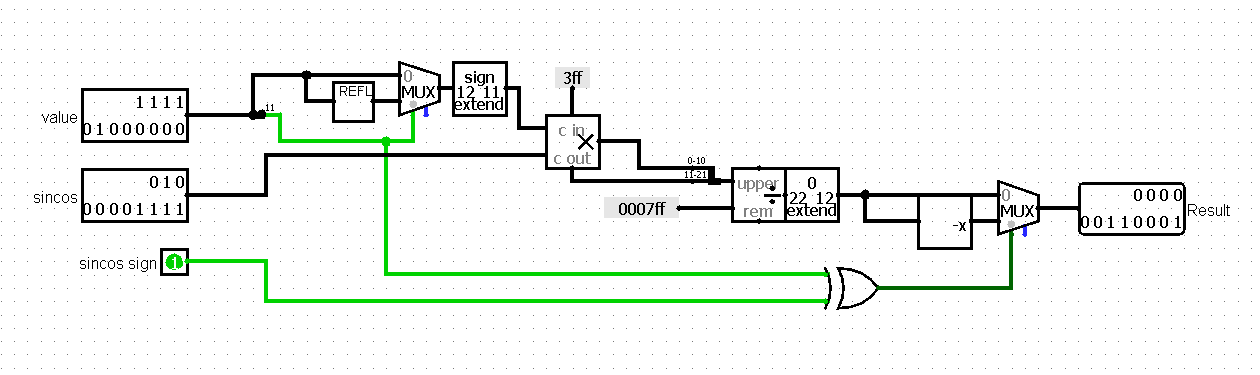
Let’s now assume that is half of the central angle for that arc. is the distance between some point on the chord and it’s centre. is the angle between the central radius and the radius to that point. is a segment of the central radius that was divided by the chord. The chord’s length is 24, same as the height of the bat.



If the distance between the ball and the bottom of the bat was , the and its sin and cos were calculated for . It is done via a Python script and saved as image for Logisim ROM.

Everything is calculated for the left bat. For the right bat sin changes its sign and cos remains the same, it is done by a scheme.

Since Logisim’s multiplication module doesn’t work well with negative numbers, the scheme for multiplication will be bigger. It will take the numbers’ absolute values and negate the result in the end if needed. vx and vy are 12-bit values, so the absolute value is 11-bit. sin and cos are stored as 12-bit values, where the first bit is the sign and an absolute value between 0 and 2047 represents fractional value from 0 to 1. Result of the multiplication we get 22-bit number, then it is divided by 2047. Since integer division truncates the result, the result and therefore the overall speed of the ball will get lower and lower. To prevent this, 1023 is added before the division.

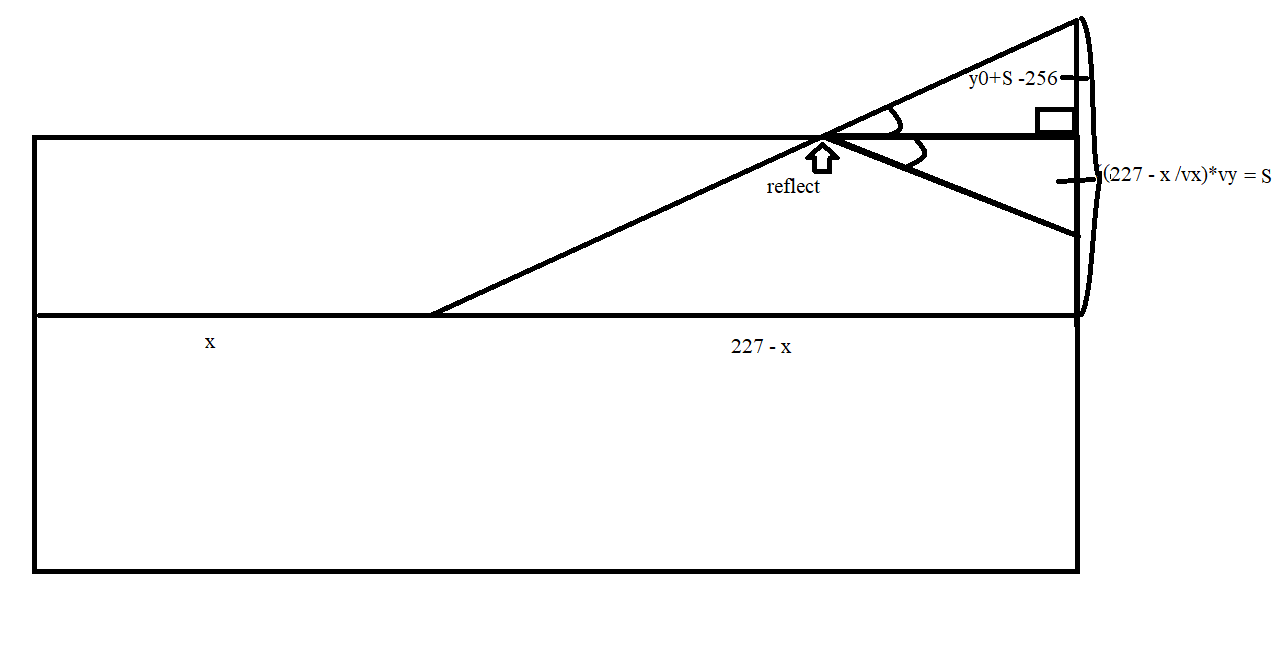


This design may cause situations when vy is much bigger than vx, and the ball bounces almost vertically. Another scheme, when vy is more than 4 times bigger than vx, puts predefined values instead of broken velocities. These values exactly differ in 4 times.

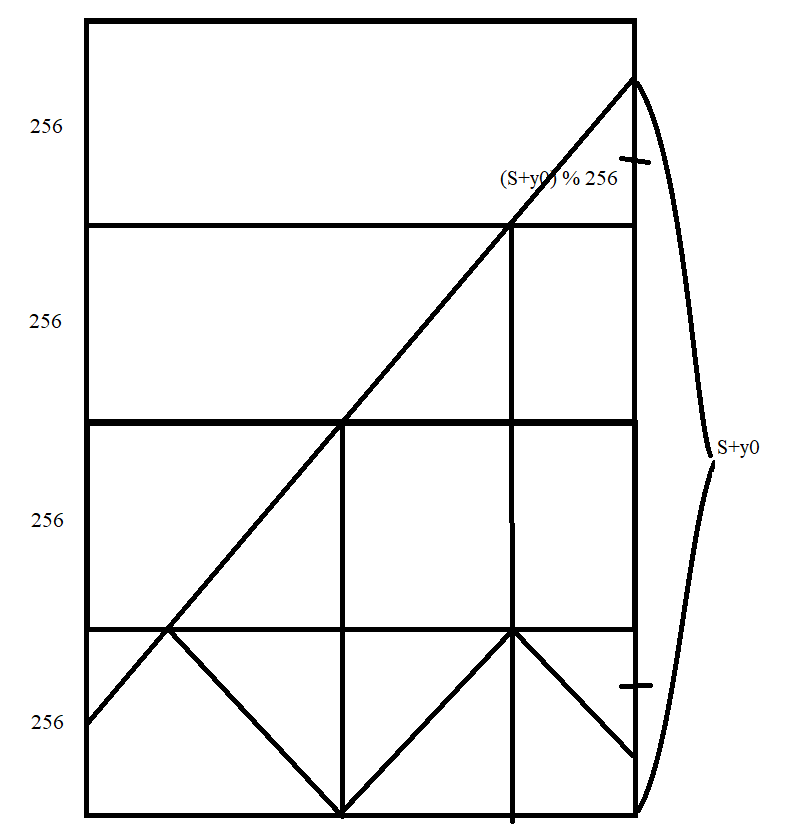
Calculating position for right bat

Below is the algorithm for the CdM-8 written in pseudo code. When the speed of the ball is positive, which means that the ball is moving right, the main calculation begins. The summary is that we first calculate the distance between the right racquet and the ball, which is 224 -xball. We add 3 to that because if vy is big the ball travels too much. Then we divide the number we got by vx, getting time the ball needs to reach the right racquet column. After that we multiply it by vy to get the distance the ball will travel along the Y axis. If vy is negative we make it positive. While multiplying it, we count carry bits, which indicate that the ball has bounced off the horizontal wall. If vy was negative we negate the y coordinate of the ball before adding it to our number. If count of carry bits is odd, we negate the result. Otherwise it remains the same. This is the optimal coordinate of the racquet.

Y0 is the Y coordinate of the bat initially. As there was one reflection on the drawing, the result will be 512 – S – y0 instead of S+y0.



D = (S + y0) % 256 is the distance between the corner and the Y coordinate that we calculate. If the number of lines 256\*n (n = 1,2,3…) we cross is odd, the said corner is at the top of the screen, otherwise it is at the bottom. If the corner is top we subtract D from 256. If the corner is bottom we just take D as a result. The number of carry bits equals the number of lines 256\*n.



if VX <= 0:

move right bat to the center

wait while VX <= 0

get XBALL, YBALL, VY and VX

# XBALL changed while we were getting YBALL

increase XBALL by VX

XRIGHT = 227

# it is around the x coordinate of the right bat

# it works better than with the exact coordinates

get XRIGHT - XBALL

#it is the distance to the right bat from the current position

put XRIGHT - XBALL as the first operator for the external calculator

if VY < 0:

put (-VY) as the second operator

else:

put VY as the second operator

put VX as the third operator

get (XRIGHT - XBALL) \* VY/ VX from the calculator

# the 8 least significant bits are the distance to go from the floor or the ceiling

#that’s true only if our ball starts from YBALL == 0

COORD := these 8 bits

# the 8 most significant bits are the number of reflections from the floor or the ceiling

REFL := these 8 bits

add YBALL to COORD

# COORD overflows after 255

if we had an overflow:

add 1 to REFL

# now COORD is the true distance from one of those two edges

if VY was positive:

if REFL is odd:

COORDS is the distance from the ceiling

else:

COORDS is the distance from the floor

if VY was negative:

if REFL is odd:

COORDS is the distance from the floor

else:

COORDS is the distance from the ceiling

if it’s the distance from the ceiling:

COORD := 256 – COORD

return COORD as the position for the right bat to catch the ball

go back to the beginning