MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION

FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION

"NOVOSIBIRSK NATIONAL RESEARCH UNIVERSITY

STATE UNIVERSITY"   
(NOVOSIBIRSK STATE UNIVERSITY, NSU)

15.03.06 - Mechatronics and Robotics

Focus (profile): Artificial Intelligence

|  |  |
| --- | --- |
| Job topic: | **TERM PAPER**  **‘TETRIS’** |

Zaichikova Valeriia

Kravchenko Egor

Zlobin Dmitry

Novosibirsk

2024

[PAGEREF \_Toc158500730 \h3](#_Toc158500730)

[1 terms AND ABBREVIATIONS 3](#_Toc166192009)

[2 Introduction 4](#_Toc166192010)

[3 Purpose and area of application 4](#_Toc166192011)

[4 Functional characteristics 4](#_Toc166192012)

[5 Technical characteristics 10](#_Toc166192013)

[7 Sources used in developMENT 13](#_Toc166192014)

# terms AND ABBREVIATIONS

|  |  |
| --- | --- |
| CdM8 Mark5 | Coco de Mer 8 Mark5 – the processor which was made in Logisim |
| LED | Light-emitting diode |
| Tetromino (tetrominoes) | Figure for the game of Tetris, geometric shape composed of four squares, connected orthogonally |
| Randomizer | A device to generate a value from a range randomly |

# Introduction

The subject of our collaborative project is focused on Tetris, one of the earliest and most renowned video games globally, developed in 1984 by the Soviet programmer Alexey Pazhitnov. This game quickly achieved widespread acclaim internationally.

The project is based on the standard rules of the game Tetris. On a 10x20 playing field, various figures, called “*tetrominoes*”, fall from above. If figures are collected in one full line, this line disappears, and the player receives a certain number of points. The number of points awarded for each line is as follows: 1 line – 100 points, 2 – 300, 3 – 700 and 4 – 1500. If a new figure cannot be accommodated on the playing field, the game will be over. The objective for the player is to maintain their position for as long as possible or to achieve the highest score.

The project was implemented on **Logisim** with a **CdM8 Mark5** processor and writing code for it via **CocoIDE**. All of the above project tools are part of our Digital Platforms subject curriculum.

# Purpose and area of application

The objective of the program is to implement it on low-level platforms such as assembler and logic circuits. The program is designed for use in gaming and entertainment spheres.

# Functional characteristics

Сode for our project is used to generate random figures for the gameplay. The main problem with the Random Generator is that a «drought» or «flood» can happen. This means that some figure may not fall out for a long time or, on the contrary, go several times in a row. The Random Generator in our code solves these problems by using the «bag» system. In this system, a list of shapes is placed in a «bag», after which the shapes are randomly removed from it one by one until the «bag» is empty. When it is empty, the pieces return to it and the process repeats. The Random Generator has a «bag» of size 7 (7-bag), that is, a "bag" filled with each of the 7 tetrominoes.

**CdM-8 full source code for von-Neumann architecture**

asect 0x00

mainLoop:

do

ldi r3, 0xF1

ld r3, r3

tst r3

until nz

jsr checkFigures

do

jsr random

ldi r1, figures

add r0, r1

dec r1

ld r1, r2

tst r2

until nz

ldi r3, 0x00

st r1, r3

ldi r1, figure

st r1, r0

ldi r1, 0xF1

st r1, r3

br mainLoop

#generator of random numbers from 1 to 7

random:

ldi r1, seed

ld r1, r1

ldi r0, time

ld r0, r0

xor r0, r1

inc r0

ldi r2, time

st r2, r0

ldi r0, mask

ld r0, r0

and r1, r0

beq isZero

jsr getFigure

rts

#auxiliary function for random

isZero:

ldi r0, mask

ld r0, r0

rts

#auxiliary function for random

getFigure:

ldi r1, seed

st r1, r0

rts

#check of "bag" with figures

checkFigures:

ldi r2, figures

ldi r3, mask

ld r3, r3

#auxiliary function for check

checkLoop:

while

tst r3

stays nz

ld r2, r0

if

tst r0

is nz

rts

else

inc r2

dec r3

fi

wend

ldi r2, figures

ldi r3, mask

ld r3, r3

#auxiliary function for check

refillFigures:

while

tst r3

stays nz

ldi r0, 0x01

st r2, r0

inc r2

dec r3

wend

rts

#memory section

asect 0xD0

#memory allocation for figures array

figures: dc 1,1,1,1,1,1,1

asect 0xE0

#memory allocation for randomizer

time: dc 0x01

seed: dc 0x01

mask: dc 0x07

asect 0xF0

#memory allocation for CDM-8 co-work

figure: ds 1

status: dc 1

end

**Code parts’ description**

*A section of code for allocating memory for a "bag", components for a random number generator and working with the CdM-8 processor:*

asect 0xD0

#memory allocation for figures array

figures: dc 1,1,1,1,1,1,1

asect 0xE0

#memory allocation for randomizer

time: dc 0x01

seed: dc 0x01

mask: dc 0x07

asect 0xF0

#memory allocation for CDM-8 co-work

figure: ds 1

status: dc 1

The main cycle of the program, in which a request for a new figure from hardware is expected. After that, a random number is generated and it is checked whether it is available. *Next, it records which number was used and is sent to the hardware:*

asect 0x00

mainLoop:

do

ldi r3, 0xF1

ld r3, r3

tst r3

until nz

jsr checkFigures

do

jsr random

ldi r1, figures

add r0, r1

dec r1

ld r1, r2

tst r2

until nz

ldi r3, 0x00

st r1, r3

ldi r1, figure

st r1, r0

ldi r1, 0xF1

st r1, r3

br mainLoop

A function for generating a random number by using a timer and seed, as well as a mask that limits the range of numbers from 0 to 7, after which a check is performed if the number is not 0. This is a feature of the program. *At the end, the function returns the resulting number for further use by the program:*

random:

ldi r1, seed

ld r1, r1

ldi r0, time

ld r0, r0

xor r0, r1

inc r0

ldi r2, time

st r2, r0

ldi r0, mask

ld r0, r0

and r1, r0

beq isZero

jsr getFigure

rts

isZero:

ldi r0, mask

ld r0, r0

rts

getFigure:

ldi r1, seed

st r1, r0

rts

*The next part of the code checks the fullness of the "bag" and fills it again if necessary:*

checkFigures:

ldi r2, figures

ldi r3, mask

ld r3, r3

checkLoop:

while

tst r3

stays nz

ld r2, r0

if

tst r0

is nz

rts

else

inc r2

dec r3

fi

wend

ldi r2, figures

ldi r3, mask

ld r3, r3

refillFigures:

while

tst r3

stays nz

ldi r0, 0x01

st r2, r0

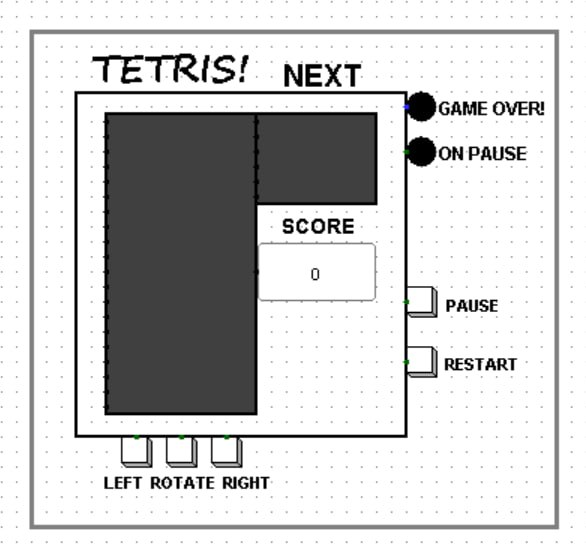
inc r2

dec r3

wend

rts

# Technical characteristics

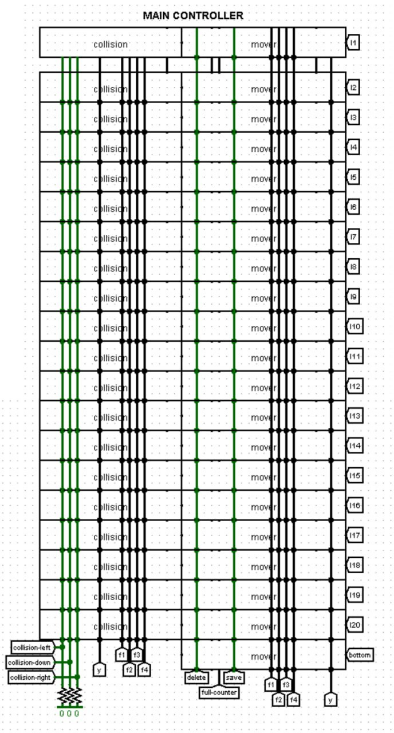
**The First section: the Output Display**

*(fig.1)* The primary output component of the program is the display layer (as previously mentioned, its dimensions are **10x20**).

On the smaller matrix **6x8**, the image of the next new output figure is printed from the top left. Additionally, buttons for controlling the game process have been implemented, including buttons for controlling the player (movement to the right, left, turn), as well as "*Pause*" and "*Restart*."

**LEDs** with corresponding signatures indicate the conclusion of the game and its current status (e.g., whether it is paused). *All components are connected to the field subcircuit.\*\*\**

*Fig.1. game ui, main circuit*

****

**The Second Section: the field**

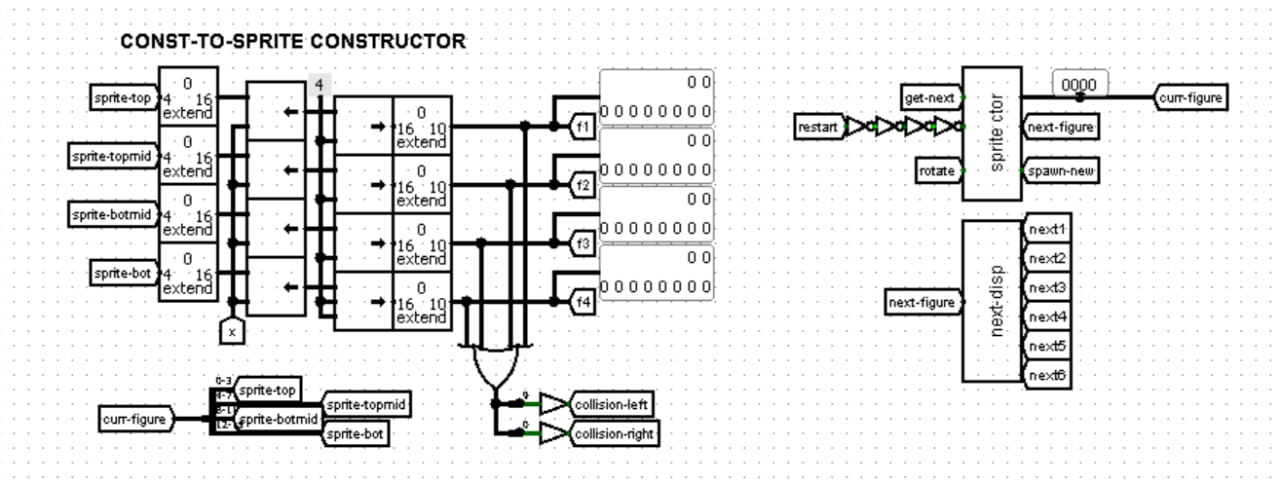
* ***The field Subcircuit: Main controller***

Output to the main display, as shown in the illustration above, is handled by the main controller block, which incorporates collision and sprite control subcircuits *(fig.2)*.

*Fig.2. the main controller*

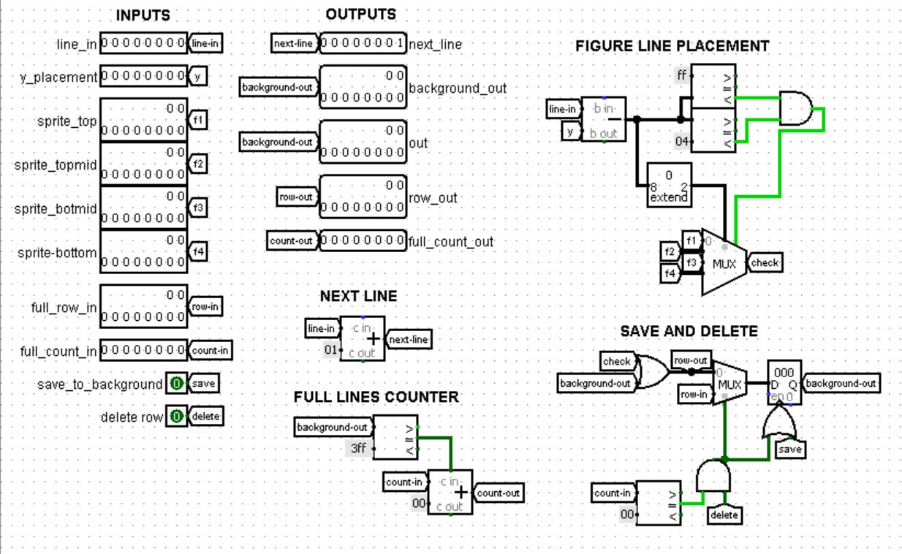
* ***The field Subcircuit: Const-to-sprite constructor***

The const-to-sprite constructor block performs sprite drawing from a pre-generated and specified constant using bit shifts *(fig.3)*.



**The Third Section: Sprite Control subcircuit**

*Fig.3. const-to-sprite constructor block*

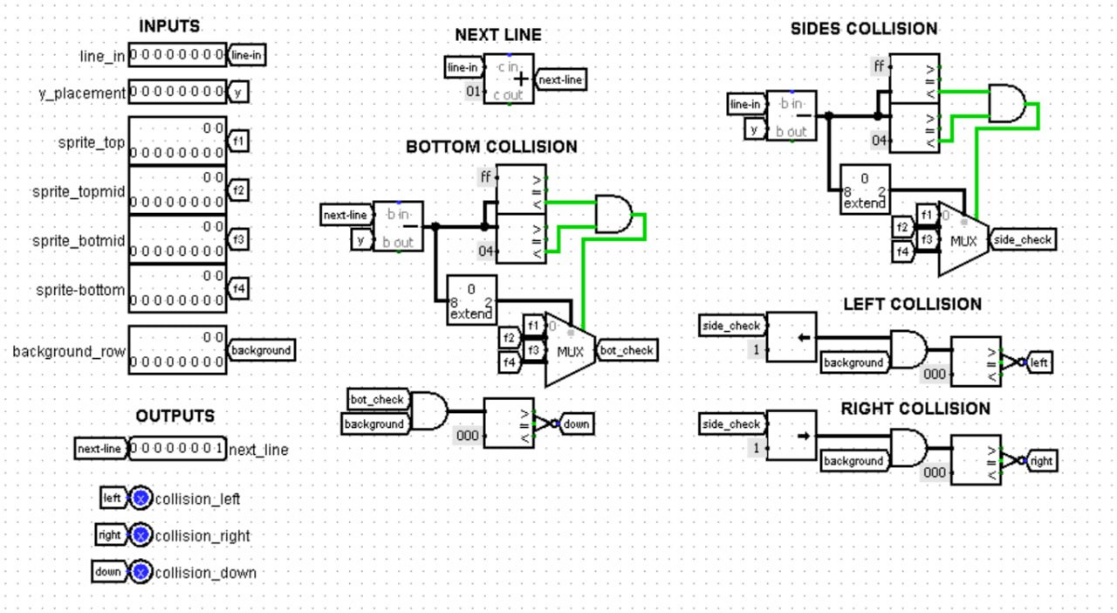


The sprite control subcircuit is responsible for controlling the position of sprites by y-position, as well as saving them to the "background" (already set figures), analysing lines for completeness, and deleting complete lines. The value of the background line is stored in a register, whose value is updated when a new shape is saved. The value of both the background and the line being changed is displayed on the playing field.

*Fig.4. sprite controller*

Lines are analysed sequentially, which allows for the correct implementation of the image shift when collecting a full line in accordance with the game rules and the instant calculation of such lines. *(fig.4)*

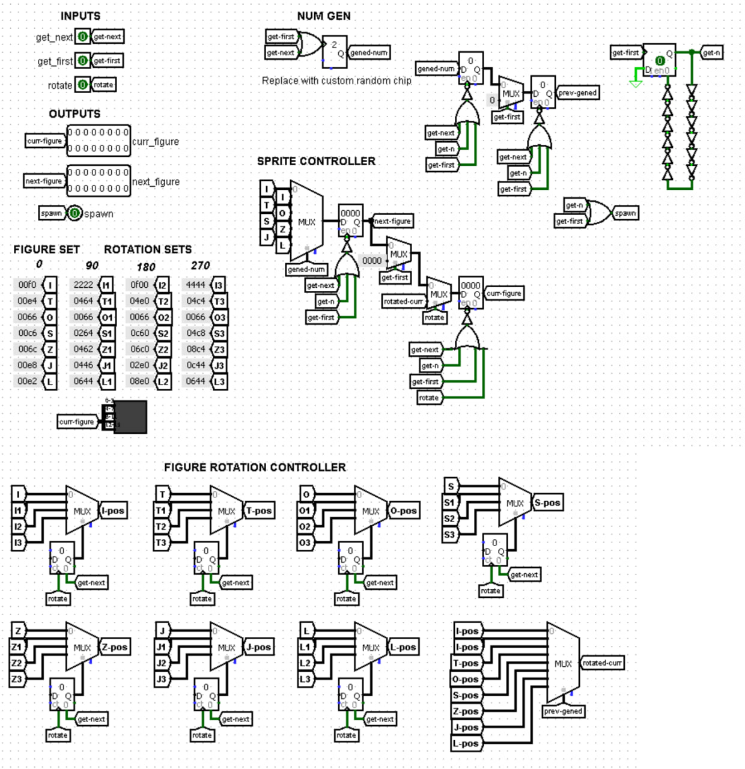
**The Fourth Section: Collision subcircuit**

****

*Fig.5. collision controller*

*(fig.5)* The collision subcircuit calculates the side and bottom collision using arithmetic operations and compares the changing value of the line with the values of the background, as well as the bottom and side walls of the playing field. The construction is similar in execution to that described above.

**The Fifth Section: Get\_Figure subcircuit**

In the get figure subcircuit, the sprite values are managed and rotated based on them. In order to ensure correct and consistent operation of the scheme on restart, a restart value delay is necessary. Furthermore, the process of displaying the sprite on an additional window with the following figure is also set here. *(fig.6)*

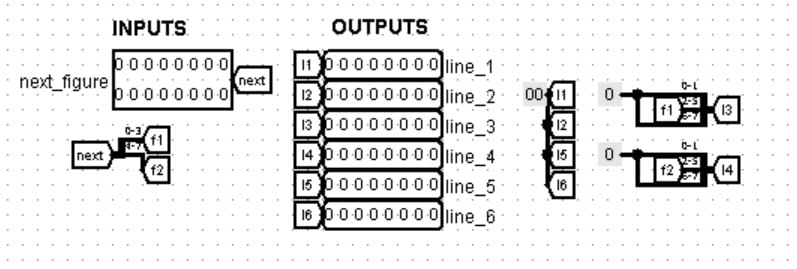
The ***get\_figure subcircuit*** is responsible for converting a number generated by the ***randomizer*** into one of the seven sprites available in the game. This is stored in the figure set block, along with the values of the numbers generated for them. These values are stored in the corresponding registers.

Additionally, counters and multiplexers are used to realise rotations based on sprite visuals replacement. This is done in a similar way to the existing implementation of the player's figure output, which allows for a cost-effective solution...

*Fig.6. get\_figure subcircuit*

**The Sixth Section: The Output on the Second Display subcircuit**

* ***The Const-to-Sprite Block’s Subcircuit: The Output on the Second Display***



The subcircuit of the output to the additional display *(fig.7)* is created by drawing with the help of constants, as it only requires the display of the figure, not its control or manipulation...

*Fig.7. second display output*

1. CONCLUSION

The project of implementing the Tetris game on assembler and through logic circuits has been successfully completed. The main goal, to demonstrate the capability of low-level platforms to create a complete gaming application, has been achieved. Functionality and stability of the program have been verified by tests, making it suitable for entertainment purposes.

# Sources used in developMENT

* <http://ccfit.nsu.ru/~fat/Platforms>
* <http://www.cburch.com/logisim/docs/2.6.0/ru/guide/tutorial/index.html?authuser=1>
* *“Computing platforms”, A. Shafarenko and S.P. Hunt, School of Computer Science University of Hertfordshire 2015*
* https://en.wikipedia.org/wiki/Read-only\_memory
* https://en.wikipedia.org/wiki/Random-access\_memory
* https://tetris.wiki/Super\_Rotation\_System
* *...*