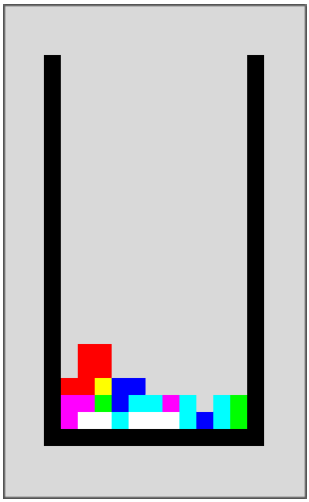
Spectris: Project Proposal

Responsibilities:
Controller: Alfredo
Game logic: Benny / (Jose/Alfredo secondary)
Video: Benny / (Jose secondary)
Audio: Jose



Overview

Our game Spectris can be divided into 3 major sections: The controller inputs, the game logic, and the audio/video outputs. Each section has core features as well as expanded features that we wish to implement for extra interest / entertainment value. The project will be constructed on a Nexys 4 board.

For the inputs to our game, the Nexys 4 buttons will work as debug / simple controls, which will including moving left, right, down, and turning. But for a more interesting player experience, we will also create a motion controller that allows the player to control the game with a hand-worn apparatus. This control glove would be viewed by a camera, which would connect to a module that converts the viewed image into game inputs.

The glove/camera apparatus is going to be able to detect movements in the player's hand in the following way: If the module detects that the player has moved his hand to the left of to the right it will map it to the direction the block should move. If the module detects that the rotation gesture has happened

then this controller module will send a signal to the game logic telling it to rotate the block in the specified direction (clockwise vs counterclockwise). If again the module detects that the send-block-down gesture has been established, then the module will send a signal to the game logic telling it to accelerate it downwards.

There will be two regs. One 3 bit reg (called gesture) and a 1 bit reg called sig.

The glove module will detect if any of the gestures have been detected. The gesture include: moved left, moved right, moved down, no movement and a rotation gesture.
If the rotation gesture has been detected, the piece will be rotated 90 degrees. If the down gesture has been detected, then the piece will move all the way down. If either left or right has been detected the controller module will start a timer and if the timer is greater than a parameter the block will move in that direction and the signal bit will be reset to 0 and wait until the player had moved in the same direction for some time (passed the time parameter). If the controller module detects no valid gesture then the block will continue to move downward as normal.

The game logic is based on a finite state machine that changes state depending on player inputs (in menus) or whether the game has ended (in game). The game outputs to the video and audio are as stated above and below. There is also a settings module that can be changed using the controller when the player is viewing the settings; when playing the game, some parameters are loaded from the settings module. These parameters include the difficulty setting (a few bits), which

changes things such as falling speed and the color-blind setting (one bit), which will cause stripes to be drawn on blocks depending on their colors. Upper bounds: 10 settings, each 3 bits wide at most.

The FSM randomly chooses which Tetris block to use, and takes in how fast the blocks are falling, and whether a rotation or translation is occurring. In return, the FSM will update the current falling block's position, performing any translations and rotations if needed.

The tetris game board will be 10 blocks long by 20 blocks high, as per a normal tetris board. Each block is fully filled by a single color, so 200 RGB values must be stored. Memory used will be on the order of 10kB for full RGB values. However, for the pure basic version in which only three colors are RGB, each block's color value can can be simplified to three bits, representing each of Red, Green, and Blue. One possible way to draw the blocks onto the screen is to use a modified version of the blob module from lab 3.

Falling tetrominoes will collide based on their color. For example, two red blocks cannot go through each other, so if a falling red tetromino collides from falling anywhere on a red block, that tetromino will lock into place; the next tetromino will spawn and begin falling. A red block and green block will overlap to make yellow, which can then overlap with a blue block to make white (red + green + blue). Only a full line of white blocks (sum of all available colors) can clear a line. As per normal tetris, the game is over when any blocks reach the top of the screen.

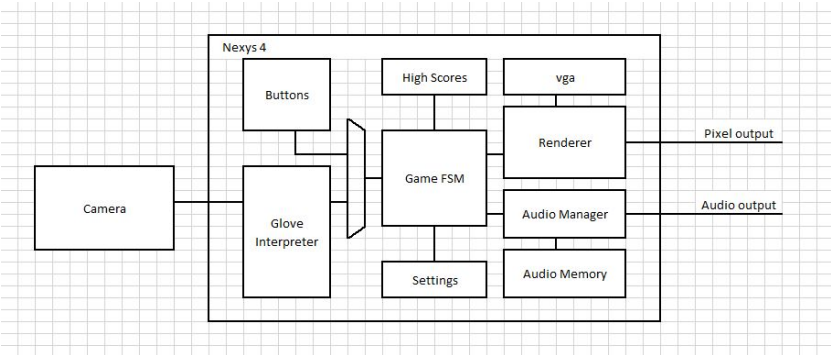
All of the game logic and settings can be troubleshooted / debugged / tested using the Nexys 4 and VGA connected monitor.

The audio will be synthesized using bitcodes saved in memory, rather than played from a stored waveform. Every certain number of cycles, the audio module loads and executes a bitcode, which will turn a note on or off at a certain pitch for a particular channel. The game FSM determines which background music should be playing; it also determines when sound effects need to be played, at which point the bitcode for the sound effect will take precedence over the background music. Each sound effect only uses 1 channel, however, so most of the music will not cut out when a sound effect plays.

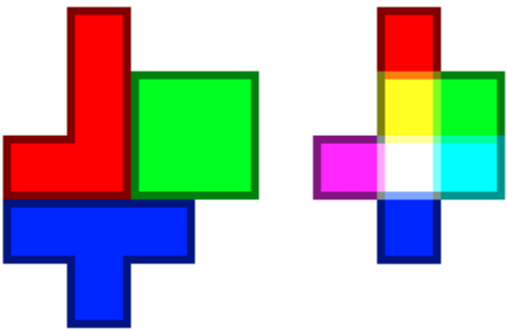
The audio module will have 4 channels: 2 square waves, 1 triangle wave, and 1 sawtooth wave. These channels, which are either off or generating a wave of a certain frequency, are summed together and outputted to the audio out of the Nexys 4.

The music and sound effects will be composed externally, then converted to bitcodes for the game. This will allow us to have higher quality audio without spending memory on audio samples. Each bitcode is 14 bits in size: 2 bits for which channel, 1 bit for on/off, 7-bit pitch value, 4-bit volume value. If the audio system reads bitcodes at a frequency of 32 Hz, then 1 minute of audio data for all 4 channels takes up 13.44 kB. For reference, 1 minute of 16 bit PCM data sampled at 44.1 kHz would require about 5.292 MB of space.

There will be at least two different choices for background music, and sound effects for cleared lines, locking tetrominoes, game over, and menu selections.



Above: The basic block diagram for our project.



Above: An example of how the colored blocks overlap. Here the blue block moved up 1 and the green block moved left 1.

Commit Goals:

- Functioning spectris, complete with generic tetris controls (Nexys 4 buttons), and block collision / line clear based on color.
- All basic controls using Nexys 4 controller.
- 2 background music tracks and 3 sound effects

Goals:

- Glove gesture controller
- Keeping Score, displayed as a number on the side next to the tetromino queue (most likely to be # of lines cleared.
- List future blocks (tetromino queue).
- Main Menu with settings for Difficulty and variable fall speed.
- Hold function, where the player can hold a piece for later.
- Glove Control Scheme
- Accelerated music when the player is close to losing at the top

Stretch Goals (try to implement some, but probably not all):

- Colorblind Mode / alternate color schemes
- Options to choose movement speed (default will be reasonable)
- Options to choose number of colors (for example 2 colors could be grey+grey = white, or red + cyan = white, as opposed to the standard red + green + blue = white)
- Gradually increasing difficulty, along with score manipulation, possibly different scoring based on more lines cleared at a time as well as difficulty
- Fancy intro screen animations.

- All block edge logic for whether a piece is still falling or landed with the rest of the stationary blocks, e.g. grey border for moving blocks, and the entire ground has a white border around it.
- Highscores List