Tetris Duel

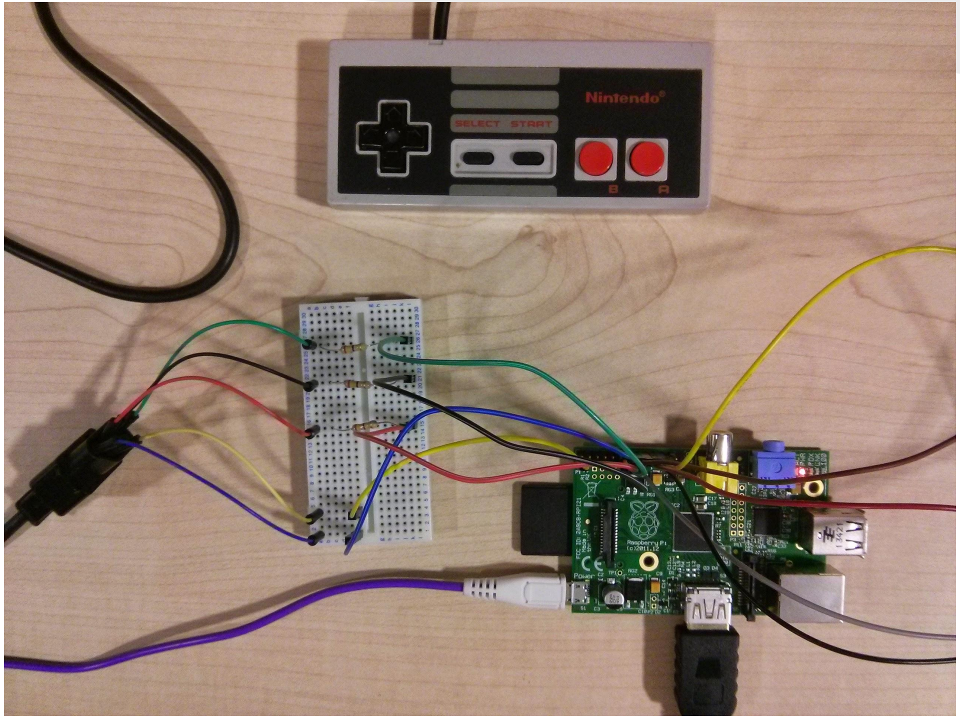
*Piotr Chabierski, Han Qiao,
Michał Sienkiewicz, Utsav Tiwary*

Overview

- Code base (lines)
  - C - 1800
  - ARM assembly - 4000
  - Java - 70
  - shell - 25
- Game assets - 6.2 MB
- Gitlab - 380 commits

Components

- 2 Raspberry Pis
- 2 NES Controllers
- Additional LEDs
- Additional parts
  - jumper wires
  - resistors
  - breadboards



Emulator

- Functional style of programming - no global variables
- Macros and inline functions to boost efficiency
- Separation of concerns - code split into 3 files, each related to a different layer of the program (easier, isolated debugging)

Assembler

- Perfect hash table for opcode decoding
  - function pointers
  - gperf to generate hash function
  - benefit: O(1) decoding, no wasted memory
- Resizable array for label storage
  - custom hash table implementation

Assembler

- Extended instruction set
  - push, pop, bl
  - benefit: shorter, clearer code
- Error reporting
  - invalid branch labels
  - immediate value too large for rotated 8 bits
  - benefit: catch programming blunders

Assembler

- Stylistic improvements
  - handles comments, indentation
  - benefit: increased readability of code
- New feature!
  - .incbin
  - benefit: import external binary (game assets)

Testsuite

- Additional test cases
- Expected output from official assembler
  - arm-none-eabi-as
  - arm-none-eabi-objcpy
- Linked via git submodule

Tools

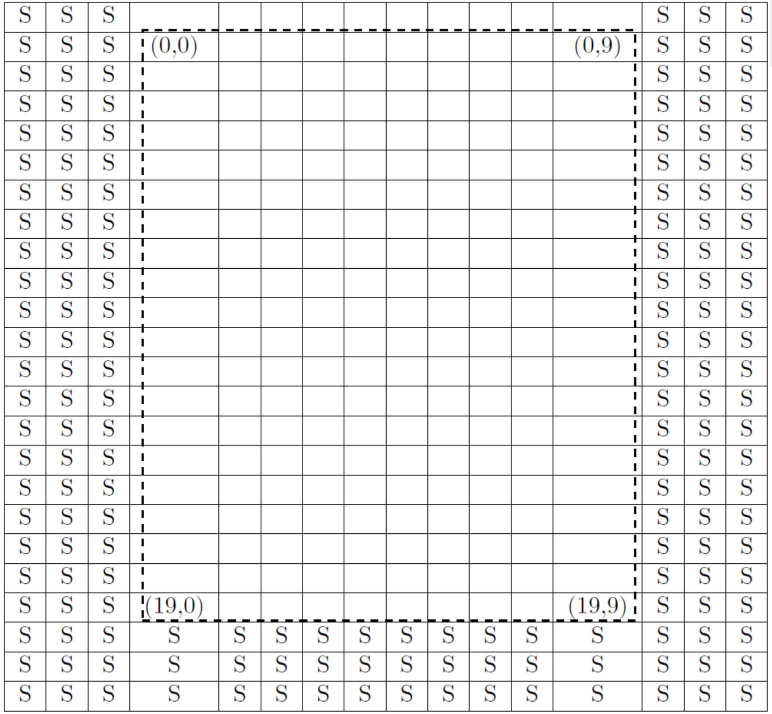
- Hex Fiend
  - compare binaries from official assembler
- Preprocessor
  - replace named variables
- Shell scripts
  - automate repetitive tasks
- Code generator
  - routines that initialize game state

Extension Overview

- Game Design
- Output - Graphics
- Main Game Loop
- Input - NES Controller
- Networking - GPIO

Game Design - Game Board

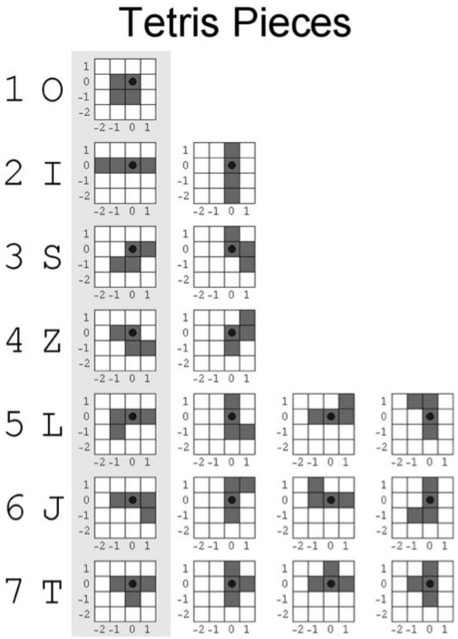
- Game Board - 16 x 24 array representing the 10 x 20 game board and **sentinel values**
- Stored in the memory as a sequence of values indicating whether the field is occupied by a piece or not
- **Sentinel value** - shows that a field is permanently occupied



S	S	S											S	S	S
S	S	S	(0,0)									(0,9)	S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S											S	S	S
S	S	S	(19,0)									(19,9)	S	S	S
S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S

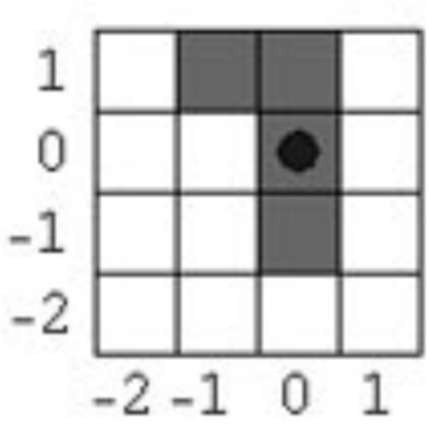
Game Design - Rotations of Pieces

- Array of Rotations - stores all possible rotations of each piece
- Each shape has a specific colour assigned to it
- Every rotation is encoded using 16 values, each indicating both whether the current field is occupied by a piece and what colour does the piece have



Game Design - Global Variables

- Game Board
- Array of Rotations
- Shape and rotation of the currently falling piece
- Shape of the next piece
- Address of the frame buffer
- Address to the temporary buffer (used in image rendering)

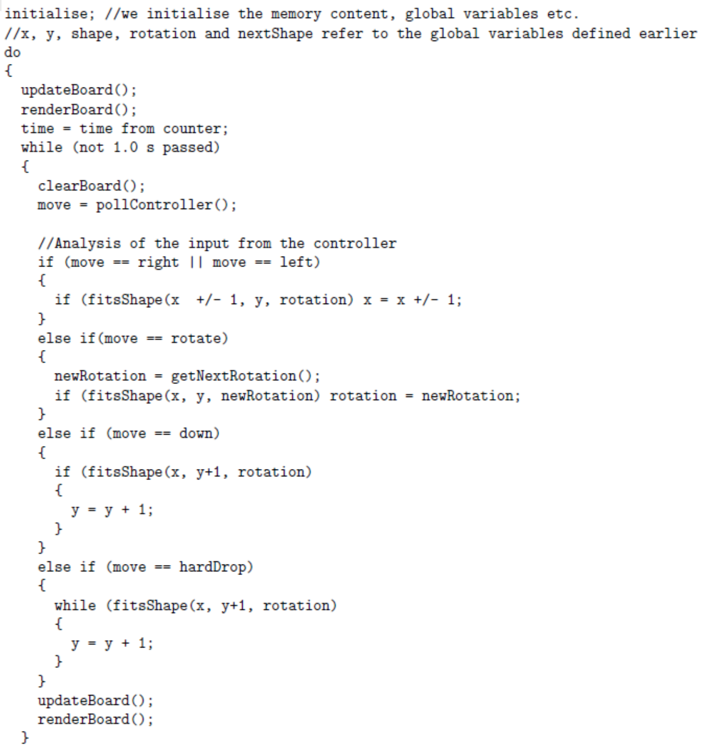


Output - Graphics

- Frame buffer negotiation
  - communication via a mailbox
- Graphics generated pixel by pixel in the temporary buffer
- Double buffering
  - smoother graphics
  - clearer code

Main Game Loop

- Whilst the time passed is less than 1 second :
    - Poll pressed buttons and move/rotate/hard drop if possible.
    - Collision detection - use of sentinel values
    - Game piece movement - simple loops thanks to our game state representation

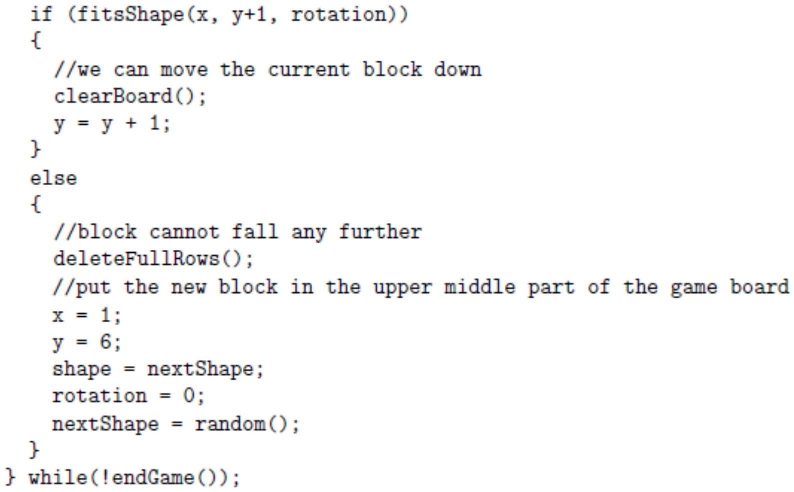


```
initialise; //we initialise the memory content, global variables etc.
//x, y, shape, rotation and nextShape refer to the global variables defined earlier
do
{
  updateBoard();
  renderBoard();
  time = time from counter;
  while (not 1.0 s passed)
  {
    clearBoard();
    move = pollController();

    //Analysis of the input from the controller
    if (move == right || move == left)
    {
      if (fitsShape(x  +/- 1, y, rotation) x = x +/- 1;
    }
    else if(move == rotate)
    {
      newRotation = getNextRotation();
      if (fitsShape(x, y, newRotation) rotation = newRotation;
    }
    else if (move == down)
    {
      if (fitsShape(x, y+1, rotation)
      {
        y = y + 1;
      }
    }
    else if (move == hardDrop)
    {
      while (fitsShape(x, y+1, rotation)
      {
        y = y + 1;
      }
    }
    updateBoard();
    renderBoard();
  }
```

Main Game Loop

- After 1 second has passed:
  - if possible, move the piece down
  - else
    - search and clear full lines
    - enter the next piece into the game board
    - get a new random next piece
- Check if the game has ended - if not, set timer to 0 and re-enter the main game loop



```
  if (fitsShape(x, y+1, rotation))
  {
    //we can move the current block down
    clearBoard();
    y = y + 1;
  }
  else
  {
    //block cannot fall any further
    deleteFullRows();
    //put the new block in the upper middle part of the game board
    x = 1;
    y = 6;
    shape = nextShape;
    rotation = 0;
    nextShape = random();
  }
} while(!endGame());
```

Main Game Loop - Multi-player

- Sending cleared lines to the opponent
- Adding received lines only after current piece collides
  - fair game - a user could be in the process of clearing several lines
  - faster, simpler code - function adding lines called less often
- Instant game over if received lines do not fit

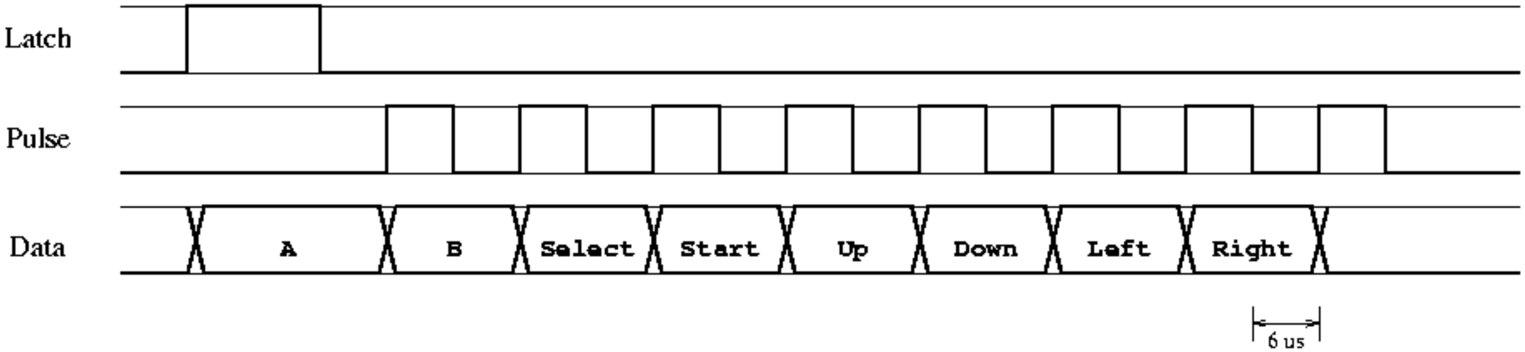
Input - NES Controller

- 8 buttons, 1 shift register

Pin Name	Pin No.	GPIO No.	fsel select	shift
Power (3.3 V)	1	-	-	-
Latch	11	17	4	21
Clock	12	18	4	24
Data	13	27	8	21
Ground	14	-	-	-

Input - NES Controller

- Polling logic
  - 1 latch signal
  - 7 consecutive clock pulse
  - 60 us cycle

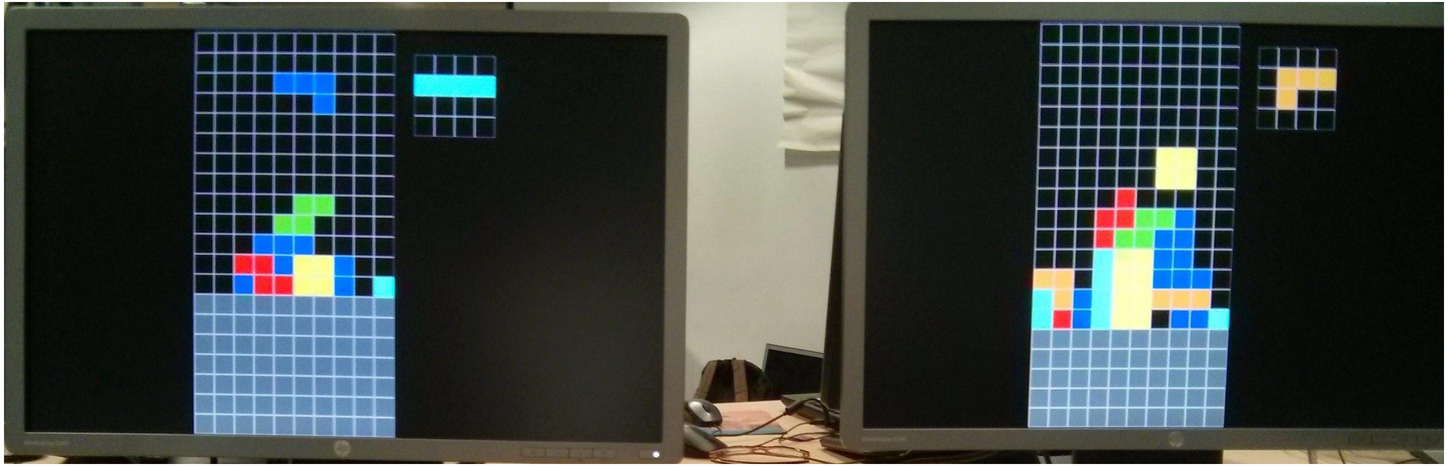


Input - NES Controller

- Optimized PollController
  - flattened all function calls
  - benefit: more responsive controls
- On-device debugging with LED
- Multimeter

Networking over GPIO

- Synchronised game start/over
- Sending/receiving lines from opponent



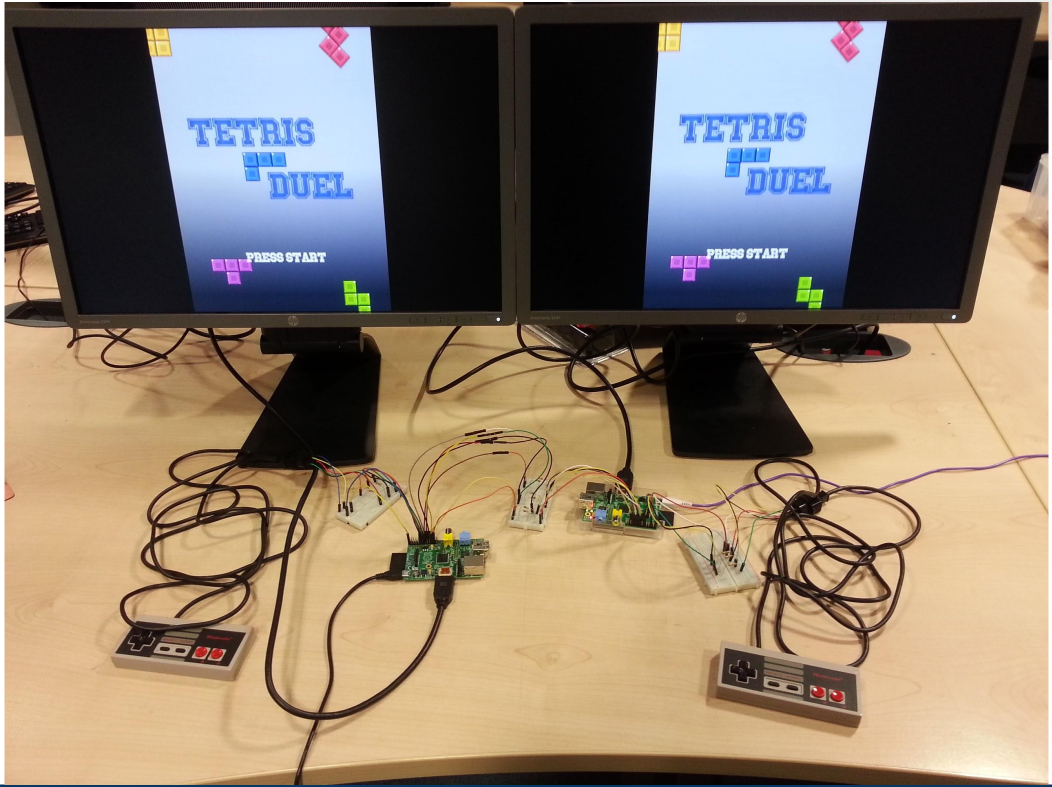
Networking over GPIO

- Receiver informs the sender that he has received a line
- Asynchronous, state dependent
  - minimizes risk of data loss
  - more responsive gameplay

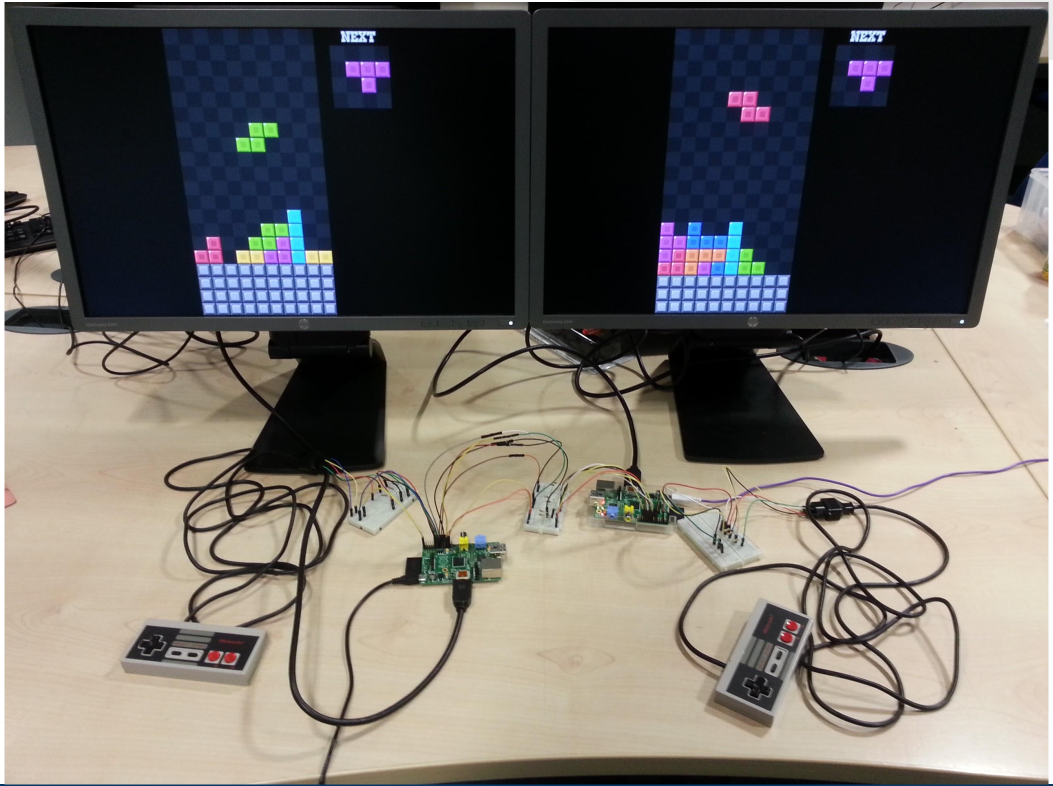
Game Demo

Tetris Time!

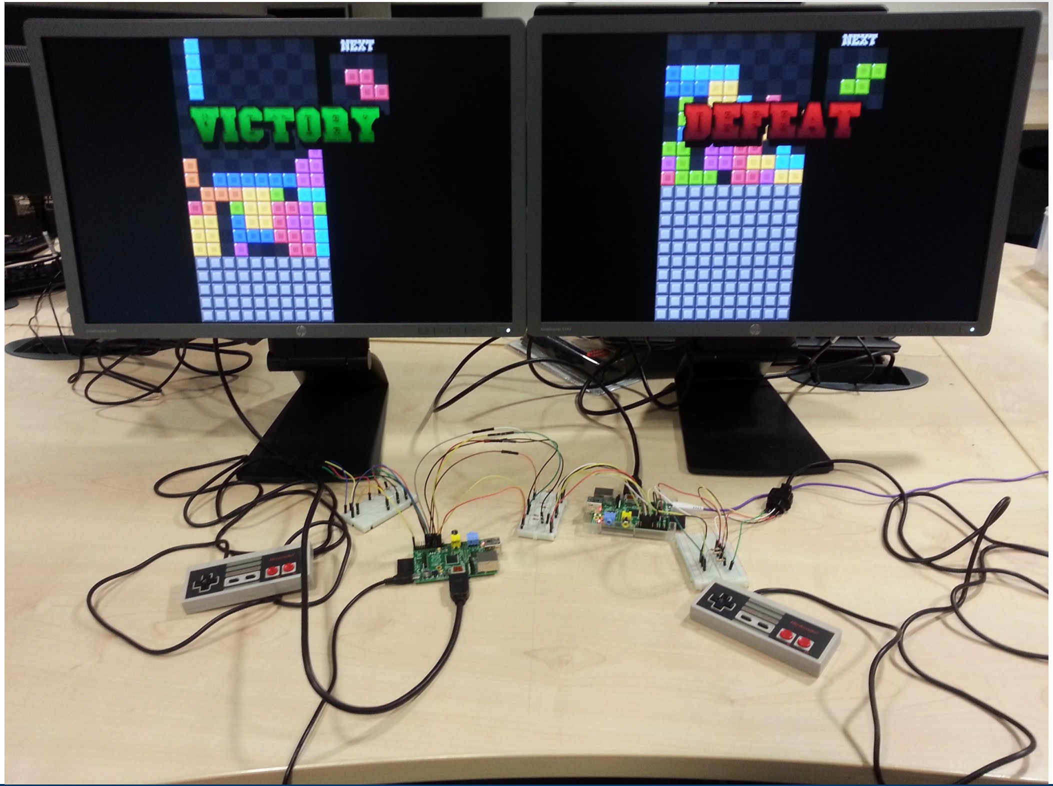
New Graphics



New Graphics



New Graphics



Limitations and Future Work

- More robust networking
- Background music
- Saving high score to SD card

Reflection

- Coding in assembly
  - tedious to work in low level code
  - difficult to debug
- Programming in group
  - Trello
  - Git branching
- Timeline
  - week 1: Part I - Part III
  - week 2-4: Extension

VICTORY