Final Report

Piotr Chabierski, Han Qiao, Michał Sienkiewicz, Utsav Tiwary

June 13, 2014

1 Introduction

For our extension, we decided to try something highly ambitious, a multiplayer Tetris game written solely in ARM assembly and compiled with our own assembler. We got our inspiration from Tetris Battle[1], an online game that pitches Tetris players against each other and gives this classic game a more competitive aspect. Our code can be viewed in the file tetrisToProcess.s which is used to generate the final code by running our custom preprocessor.

2 Group Organisation

As our team completed Part I - Part III two days before the checkpoint, we began working on the extension immediately after submitting our checkpoint report.

During our initial meeting, we realised that we understood too little about ARM assembly to foresee all the potential problems in implementing our Tetris game. Knowing this, every member conducted extensive research independently before the next meeting. In our next meeting, everyone was well prepared so we brainstormed for all the features that we wanted to include in the final game and discussed whether they were doable within the limitations imposed by our own assembler. After coming up with a pseudo-spec that described most of the game logic in pseudo code, we prioritised a list of functions on Trello and assigned them to individual members.

Piotr's interest in graphics and rendering made him a strong candidate for implementing the display of the gameboard and the Tetris pieces. Michal's precise, mathematical thinking was best suited for working with the game logic. Han's thorough research on getting input from the controllers meant that he was already ready to type up a lot of the code to receive input from the controller. Despite the fact that we didn't have a functioning game to test the networking between the Pis, Han was able to write a lot of the networking code and alternatively test it using the Pi LEDs. Utsav worked mostly on extending the assembler and partially helped Michal with the game logic. Additionally, Piotr and Michał created our supporting software (Code Generator and Preprocessor).

Even though every member had a single responsibility assigned to them, we ensured that we went over each other's codes in order to be familiar with the different parts of the game and to help with debugging. The fact that we worked as a group in the computer labs made it a lot easier to be up to date with each other's progress.

Throughout the project, we took great care to update the Trello board (see Figure 4 and Figure 5), uploading shared resources and moving completed cards to the weekly "done" sections. These efforts paid off as it kept the whole team on track and constantly informed us of our latest progress.

3 Testing and Debugging

Most of the testing of the game had to be done manually on the device itself which was very time consuming compared to unit testing. Our favourite method was to make the LED blink when the testing code was executed correctly. As the project got more complex, we used a few tools to reduce our programming mistakes. Towards the end of the project, we also invited other computing students to play the game to catch any edge case bugs.

3.1 Emulator

Our emulator was one of the most important tools we used as it allowed quick debugging. We utilised our emulator to run parts of our binary code and pinpoint exactly what went wrong. In combination with gdb in Eclipse, the emulator helped us correct quite a few segmentation faults caused by things like misspelled labels.

3.2 Enhanced Assembler

By extending the original ARM instruction set with three new instructions, namely bl, push, and pop, our enhanced assembler satisfies the basic requirements for parameter passing and function calling. This allowed us to code in a more functoinal style and hence resulted in greater code reuse and reduced likelihood of bugs.

A separate test suite was forked from the original ruby script to test the correctness of our enhanced assembler's output. The expected output was produced by compiling the same assembly instruction using the official arm-none-eabi-as[2] and then stripping away the ELF headers with arm-none-eabi-objcpy. The new testsuite is linked to our main repository as a git submodule.

To further improve code clarity, our enhanced assembler also handles comments, indentation, and shows any invalid branching labels.

3.3 Preprocessor and Code Generator

In order to improve our efficiency and limit bugs in the code, we have created two supporting programs. The first one is our preprocessor, a Java application that replaces our variable names with numbers provided in a separate file. Although we hoped that this would allow us to imitate using variables, we soon realised that large constants would be stored by our assembler at the end of the binary file giving us a bound for the number of code lines (those constants would be accessed using an offset with upper bound of 4k lines). This made the preprocessor hardly usable, though, when possible, we still sometimes apply this method. We used much more extensively our code generating software written in C. As we needed thousands of instructions to initialise all our data structures, this tool greatly improved our efficiency.

3.4 Custom Shell Scripts

Using these scripts to perform manual and repetitive tasks increased our productivity and team morale.

assemble.sh compiles our game with both official and custom assembler.

transfer.sh copies kernel.img to SD card and ejects the SD card.

3.5 Third Party Tools

Hex Fiend used to compare the compiled binary of our Tetris game to the output of the official assembler. We were happy to find out that none of the current 8000 lines of assembly instructions in our Tetris game differed from that of the official assembler output. Knowing this fact allowed us to eliminate incorrect assembling as a potential source of bugs in our game code.

Multimeter borrowed from Computing Support Group to test the output voltage of the pins we used to ensure that our code toggled the pins correctly.

4 Game Logic

Design of the game logic was the most crucial part of our extension as this part is directly responsible for the efficiency and robustness of our Tetris game. Due to the fact that we implemented the entire game using only a subset of ARM11 commands, we made some design decisions that in other circumstances could have appeared not optimal. However, keeping the implementation of the game logic simple and elegant was our priority. One of the main challenges that we faced was coming up with a representaion of the game state which would enable relatively easy handling of data.

4.1 Representation of Game State

The standard Tetris game board has dimensions 10×20 fields. The game board is stored in memory as a sequence of 32 - bit values, each corresponding to a field on the game board. If a value is equal to 0, it means that the field is free, otherwise it is occupied by a Tetris piece. In our implementation, values of the fields denote their colours decoded using a 32 - bit colour palette.

In order to facilitate the implementation of the functions managing the game logic, the board was surrounded by sentinel values (arbitrary values different than 0). Three columns of sentinel values were added on each side, three rows below and one empty row (unoccupied fields) above the original game board. Our choice of the number of sentinel rows or columns was motivated by the shapes and rotations of the game pieces; we wanted to ensure the proper behaviour of the game in any possible situation. Therefore, after the inclusion of the sentinel values our game board had dimensions 16×24.

In Tetris there are seven different shapes of pieces. In our implementation, for simplicity, each piece was associated with its colour and every time we wanted to indicate that a field on the game board was occupied by a given piece, the corresponding colour was used to mark the field as reserved.

In the ordinary game of Tetris, each piece has up to four possible rotations. In our program, in order to treat all shapes uniformly, we assumed that every piece has exactly four possible rotations and in the case it had less, the rotations were repeated in a cyclic manner. The rotations of each shape were stored as 4×4 squares represented by 64 bytes in memory.

In our implementation we also had to keep track of the currently falling Tetris piece. For that purpose, four global variables recording the characteristics of the piece were used, namely the x and y coordinates of the top left corner of the 4×4 square enclosing the piece, its shape (number from the range $0 \ldots 6$) and rotation (number from the range $0 \ldots 3$). Although using global variables is considered to be a bad programming practice, this decision corresponds to our idea of keeping the design simple, as then every helper function responsible for managing the game logic had easy access to the required data.

4.2 Main Game Loop

Though our pseudo-code for the main game loop seemed fairly compact, the size of the assembly implementation slightly surprised us; chiefly because of our switch statement responsible for selecting moves. Also, we had to add some code handling interactions with the second Raspberry Pi. Nevertheless, as planned, in our main game loop we firstly polled the pressed buttons for half a second and reacted accordingly. Then we proceeded to collision detection, clearing lines (and sending them to the opponent), adding lines sent from the other user and, if needed, ending the game. Thanks to our game state representation, the functions carrying out the operations such as clearing or adding a piece to the game board could be done in a quite straightforward manner. In short, we could simply subtract or add the colours representing the piece and that way we were able to move it across the game board.

4.3 Graphics Rendering

Graphics rendering was the first major milestone of our project because without graphics on the screen, it would have been impossible to debug the rest of the game logic. The initial challenge we had to overcome was communicating with the graphics processor. After some research, we learned that we needed to use special memory locations to firstly send the description of the frame buffer we wanted to obtain and then, once a positive response was received, transfer our frames pixel by pixel.

In the implementation of the rendering of the game board a technique of double buffering was utilised. The rendered image was firstly composed in a separate memory location and transferred to the frame buffer once it was completed. Applying this technique allowed more flexibile handling of the graphics and made the process of rendering smoother.

4.4 NES Game Controller

We chose to use a retro Nintendo Entertainment System controller as the input device for our Tetris game. Due to the popularity of this device, we managed to find several well documented GPIO drivers written in C for Raspbian OS[35]. But to our surprise, directly translating C code to our reduced ARM instruction set was not enough to achieve a fully functional controller - only button A worked.

After a full day of research and debugging, we pinpointed the problem to the clock pulse not generating enough voltage to drive the shift register in the controller that stored the button states. As a result, only the first button was read. To correct this issue, we swapped out the original 10k Ω resistors for 260 Ω resistors and the increased voltage enabled accurately polling of all button states. Our final wiring diagram is shown in Figure 2.

On the software side, polling is done by first sending a latch signal to the controller to store the current states of all buttons and then sending 7 consecutive clock pulses to push the next bits in the shift register out to the data line to be read[4].

As PollController is one of the most frequently called routines in our game, we decided to optimize it by flattening all function calls within this routine, such as Wait and SetPinHigh, to reduce any overheads incurred when branching into other functions.

We measured the performance gain of our optimization by setting up a routine that repeatedly toggled the LED when a button was held down. With the faster execution of PollController, we expected the delay between each button press to decrease, resulting in a higher frequency of LED toggling and hence a brighter LED. Our test result showed a noticeable (by the human eye) increase in brightness.

4.5 Networking over GPIO

We initially planned to use the established Serial Peripheral Interface pins for multiplayer gameplay as these pins enable serial data communication between devices which we believed would reduce the number of wires needed as compared to a parallel implementation. But we soon realised that the protocol was too robust for our needs.

Inspired by the controller polling mechanism found on NES consoles, we decided to roll our own custom networking protocol with only four GPIO pins, two for input and two for output. However, our first iteration revealed significant synchronization problems as, more often than not, the incorrect number of bits were polled across the network during multiplayer gameplay.

Instead of going down the hard way of synchronizing the clock timings of two Pis, we came up with an original asynchronous state dependent solution. The basic logic can be explained using a signal state table as shown in Table 1.

Pi	Pin	S_0	S_1	S_2	S_3	S_0
1	Data_{out}	low	high	high	low	low
	$\text{Status}_{\text{in}}$	low	low	high	high	low
2	Data_{in}	low	high	high	low	low
	$\text{Status}_{\text{out}}$	low	low	high	high	low

Table 1: Pi_1 sending a line to Pi_2 using a state dependent method.

In our setup, the Data_{out} and $\text{Status}_{\text{in}}$ pins of Pi_1 are wired to the Data_{in} and $\text{Status}_{\text{out}}$ pins of Pi_2 respectively. On every game loop, both SendLine and ReceiveLine are called and exited as soon as possible.

If there is a line to be sent, the sender checks the current state of $\text{Status}_{\text{in}}$ and uses that input as a switch, setting the next state of Data_{out} to high (S_1) when the current state of $\text{Status}_{\text{in}}$ is low (S_0) and vice versa.

On ReceiveLine, the receiver checks the current states of both Data_{in} and $\text{Status}_{\text{out}}$ to determine the next state of $\text{Status}_{\text{out}}$, in particular,

- If Data_{in} is high and $\text{Status}_{\text{out}}$ is low (S_1), $\text{Status}_{\text{out}}$ is set to high (S_2).
- If Data_{in} is low and $\text{Status}_{\text{out}}$ is high (S_3), $\text{Status}_{\text{out}}$ is set to low (S_0).
- In all other cases, $\text{Status}_{\text{out}}$ remains unchanged.

The state of $\text{Status}_{\text{out}}$ (or $\text{Status}_{\text{in}}$ from sender's perspective), effectively, informs the sender that the receiver has received a line and it is okay to send the next line. Hence, no time dependent synchronization is required and any risk of data loss is eliminated.

It is also worth mentioning that with this implementation, the networked Pis do not have to wait for each other to respond in order to proceed to the next instruction as it makes no difference to check the status upon next entrant of the game loop and this results in a more responsive gameplay.

5 Reflection

In our checkpoint report, we mentioned that whilst working on the assembler and emulator, our team was split up into two subteams. As the two parts (assembler and emulator) can be tested independently of each other, it was sensible to work in this manner. In retrospect, working in pairs greatly increased our productivity as we were able to complete Part I to Part III before the checkpoint.

After completing the core of the assembler and emulator, each pair knew their portion of the project inside out. In order to have an overall understanding of our collective work, we explained our code to each other and discussed what modifications could be made. This process not only improved the way we communicate our code to other programmers but also allowed us to compare and contrast different coding styles and implementation techniques so that we could improve our own programming skills. Despite the fact that we were working on different parts of the code, in the end, every member had some input into the work of the others. All in all, this approach provided the optimum condition for each team member to work in whilst maintaining the teamwork needed for the project to come together as a whole.

However, our approach whilst implementing the extension was completely different. In as integrated a program as the Tetris game, it was important that we worked on it together. Even though there were times when one member's plan clashed with another's, we settled the arguments in a cool manner and carried on with what we collectively thought to be the better idea. Had it been possible to split up the work for our extension, pair programming would have been prefered. Even though, by dividing up the work in such a manner, each person's knowledge of the program is mostly limited to their section of the code, there are very few interruptions and work is completed in a much shorter amount of time.

The organisation of our group proved to be quite efficient. It involved a lot of teamwork and code dependency. However, had any of the team members fallen behind or become lethargic, the project would have surely failed. Hence, we made sure to motivate each other to stick to the plan. At the end of the day, the success of our project was down to our team chemistry and how much we wanted this project to come through.

5.1 Collaboration using Git Flow

We started the development of the extension following the git flow model proposed by Vincent Driessen. However, as development progressed, we decided that strictly following this work flow did not bring much benefit to our project due to the following reasons.

1. The lab machines do not have git flow installed and it soon becomes too cumbersome to invoke the respective git commands manually.
2. The git flow model is designed for regular releases over a long period of development time, typically over 3 months. As a result, the benefit did not outweigh the cost of learning a new work flow.

Despite that, during the short period when git flow was actually in use, its benefit was apparent. By the end of the project, our commits were tagged with meaningful messages which clearly identified our development milestones. This is great for book keeping and future references.

5.2 Utsav's Reflection

It was refreshing, finally, to work with a group who were willing to go far beyond the spec and do something really cool with the project. Despite that, I had never found myself in such an intense working environment. The deadlines that we set for ourselves seemed impossible to me at times. In fact, I was always afraid that at some point, I was going to fall behind. However, I was glad to see that my teammates didn't allow that to happen. Even though there were days when I wasn't available, they'd fill me in on the group's progress the very next day. I perhaps didn't have as much input into the extension as the rest but looking back on it all, the experience of working in such a fast-paced group is one that will definitely prepare me well for any future projects.

5.3 Han's Reflection

With a highly motivated and driven team, it becomes difficult to pull myself away from the project. The sense of achievemet we get from accomplishing exactly what we have set out to do is immensely satisfying. Although at times we hold different opinions, our shared vision of making something awesome always brings us back together. When working in such cohesive teams, there is really no fixed role for any member as each of us is willing to take the initiative to fix problems as they arise. Simply put, we get things done. And we get things done right.

5.4 Michał's Reflection

From the very beginning, it was clear to me that we are all very driven and determined to create something impressive. One of the indicators was the fact we had bought two Pis even before the project officially began, just because we wanted to start experimenting as soon as we could. Although I was unsure how well would we cooperate, thanks to the passion and good team spirit we managed to finish the emulator and assembler in a remarkable time frame. We maintained our pace throughout the whole project, sometimes working on our game for well above 12 hours a day. I really enjoyed the challenge of handling game logic and maintaing the integrity of the code. I had to cooperate closely with Han and Piotr to make the program run smoothly.

5.5 Piotr's Reflection

Working on the project together with Han, Michał and Utsav was a very rewarding experience for me. Great enthusiasm in all team members backed by dedication and creativity enabled us to make fast progress with our work. In our case, the key to successful cooperation was skillful delegation of tasks thanks to which we could do lots of independent work simultaneously. This project made me appreciate the benefits of working in a team as discussing and brainstorming ideas together allowed us to save great amounts of time and enhanced the quality of our work. Designing the extension taught me the importance of careful planning and having a positive attitude towards your work, even if some problems seem insurmountable at first glance. Overall, thanks to the great atmosphere in the team and challenging nature of our extension, I got deeply involved in our project, which provided me both with satisfaction and a number of new skills.

6 Future Work

We plan to use the time before our presentation to enhance the gameplay, improving our welcome and exit screens, creating beautiful tiles and playing background music.

7 References

1. "Tetris Battle 2P - Free online Tetris game at Tetris Friends." 2009. 12 Jun. 2014
   http://www.tetrisfriends.com/games/Battle2P/game.php
2. "GCC ARM Embedded in Launchpad." 2011. 12 Jun. 2014
   https://launchpad.net/gcc-arm-embedded
3. "Index of /~mhiienka/Rpi/ - Niksula." 2012. 13 Jun. 2014
   http://www.niksula.hut.fi/~mhiienka/Rpi/
4. "The NES Controller Handler - MIT." 2002. 12 Jun. 2014
   http://www.mit.edu/~tarvizo/nes-controller.html
5. "NES Controller on the Raspberry Pi | Gordons Projects." 2012. 12 Jun. 2014
   https://projects.drogon.net/nes-controller-on-the-raspberry-pi/
6. "Step01 - Bare Metal Programming in C Pt1 | Valvers." 2013. 13 Jun. 2014
   http://www.valvers.com/embedded-linux/raspberry-pi/step01-bare-metal-programming-in-cpt1
7. "Computer Laboratory - Raspberry Pi: Baking Pi - Operating ..." 2012. 13 Jun. 2014
   http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/
8. "ARM assembler in Raspberry Pi - Chapter 1 | Think In Geek." 2013. 13 Jun. 2014
   http://thinkingeek.com/2013/01/09/arm-assembler-raspberry-pi-chapter-1/

8 Appendix

Pin Name	Pin No.	GPIO No.	fsel offset	shift
Data_{out}	15	22	8	6
Data_{in}	16	23	8	9
$\text{Status}_{\text{out}}$	18	24	8	12
$\text{Status}_{\text{in}}$	22	25	8	15

Table 2: GPIO pins used for networking.

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

Table 3: GPIO pins used for controller.

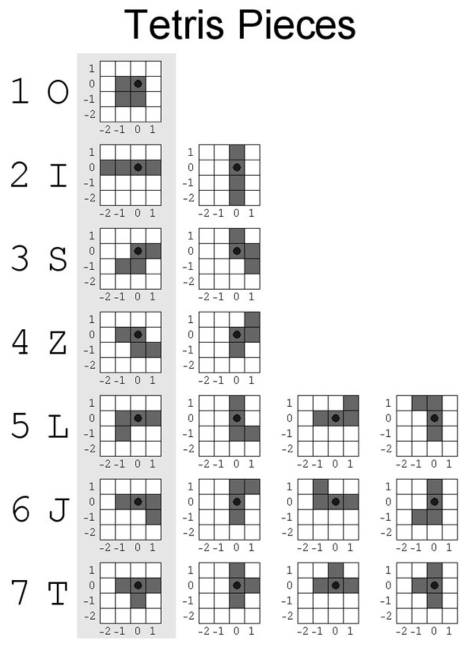


Figure 1: A 4x4 representation of all Tetris pieces and rotations. source: http://colinfahey.com /tetris/tetris.html

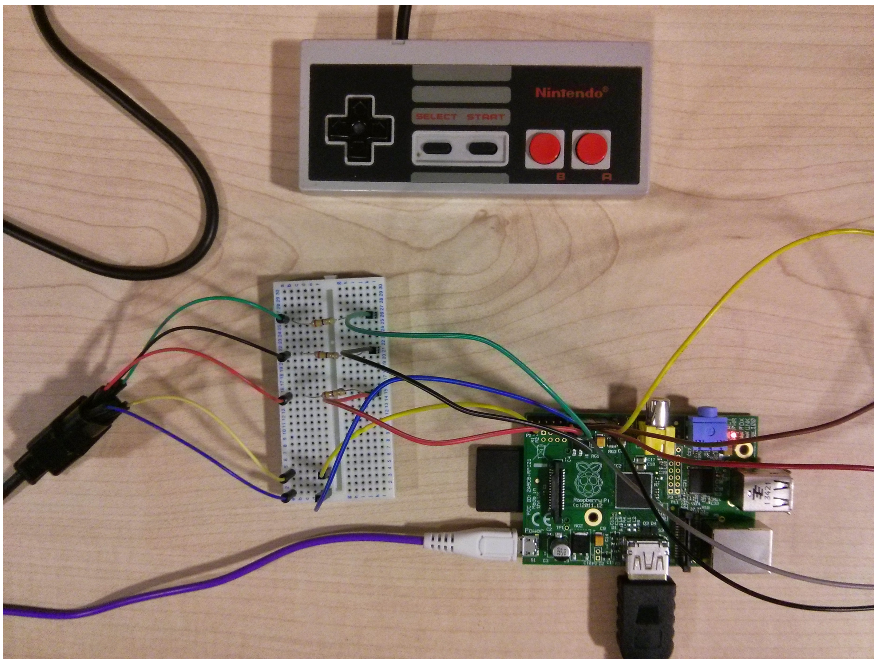


Figure 2: Connecting NES controller to Raspberry Pi using GPIO.

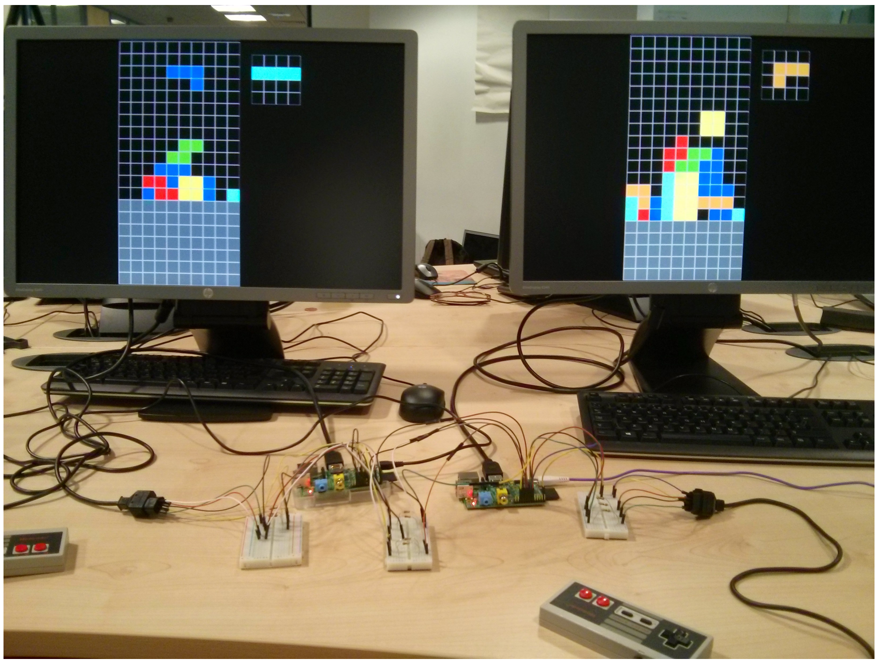
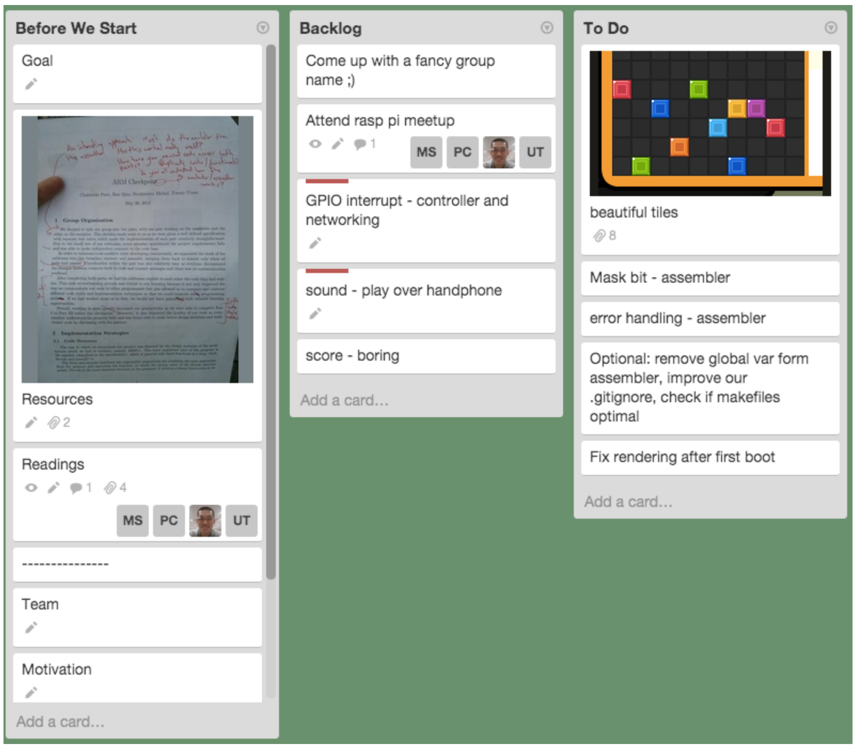


Figure 3: Actual gameplay with rendered graphics.



Before We Start

Goal

Resources

2

Readings

1 4

MS PC UT

Team

Motivation

Add a card…

Backlog

Come up with a fancy group name ;)

Attend rasp pi meetup

1

MS PC UT

GPIO interrupt - controller and networking

sound - play over handphone

score - boring

Add a card…

To Do

beautiful tiles

8

Mask bit - assembler

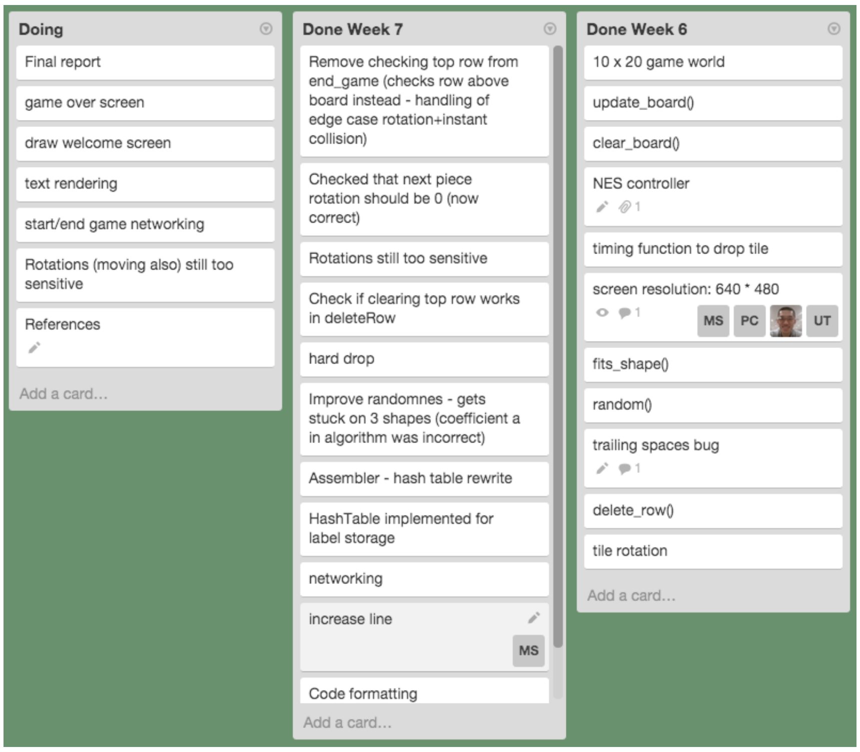
error handling - assembler

Optional: remove global var form assembler, improve our .gitignore, check if makefiles optimal

Fix rendering after first boot

Add a card…

Figure 4: Screenshot of our Trello board showing a list of tasks and backlog.



Doing

Final report

game over screen

draw welcome screen

text rendering

start/end game networking

Rotations (moving also) still too sensitive

References

Add a card…

Done Week 7

Remove checking top row from end_game (checks row above board instead - handling of edge case rotation+instant collision)

Checked that next piece rotation should be 0 (now correct)

Rotations still too sensitive

Check if clearing top row works in deleteRow

hard drop

Improve randomnes - gets stuck on 3 shapes (coefficient a in algorithm was incorrect)

Assembler - hash table rewrite

HashTable implemented for label storage

networking

increase line

MS

Code formatting

Add a card…

Done Week 6

10 x 20 game world

update_board()

clear_board()

NES controller

1

timing function to drop tile

screen resolution: 640 * 480

1

MS PC UT

fits_shape()

random()

trailing spaces bug

1

delete_row()

tile rotation

Add a card…

Figure 5: Screenshot of our Trello board showing what we are currently working on.