Assembly Project: Tetris

Yael Lyshkow
Dharma Ong

April 1, 2024

1 Instruction and Summary

1. Which milestones were implemented? We implemented all milestones. Specific features are listed below

   Easy features:

   (a) Gravity (Feature 1)

   (b) Game Over Screen + Quit + Retry (Feature 3)

   (c) Sound effects (Feature 4)

   (d) Pause feature (Feature 5)

   Hard features:

   (a) Track and display Score (Feature 1)

   (b) Play Korobeniki (Feature 5)

2. How to view the game:

   Unit width: 8

   Unit length: 8

   Display width: 256

   Display length: 256

   How to play:

   (a) Open the .asm file and press the green run button or Control + K

   (b) A blue I type tile will appear, use WASD to control its movement

   (c) Play the game

   (d) Once the game ends, a game over screen will appear. The letters R and Q will also show up. To restart, press R, to quit, press Q.

3. Game Summary:

   - Basic Tetris game (only supports I-type tiles for now)
   - Fun features and add-ons coming soon! (Score display etc.)
   - WASD controls

m

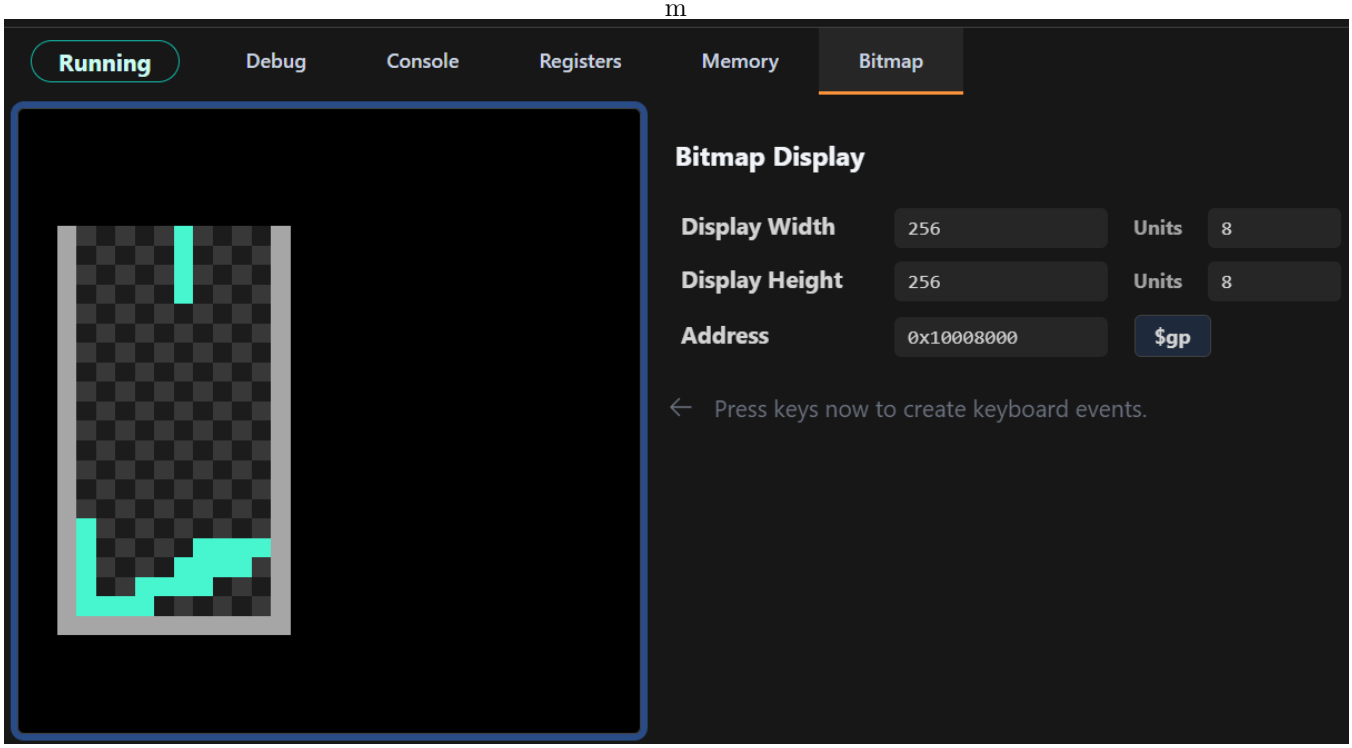

Figure 1: Sample state of the board

2 Adding music

We added music and sound effects by using the syscall function with $v0$ set to 31 (MIDI Out)

References (Korobeniki):

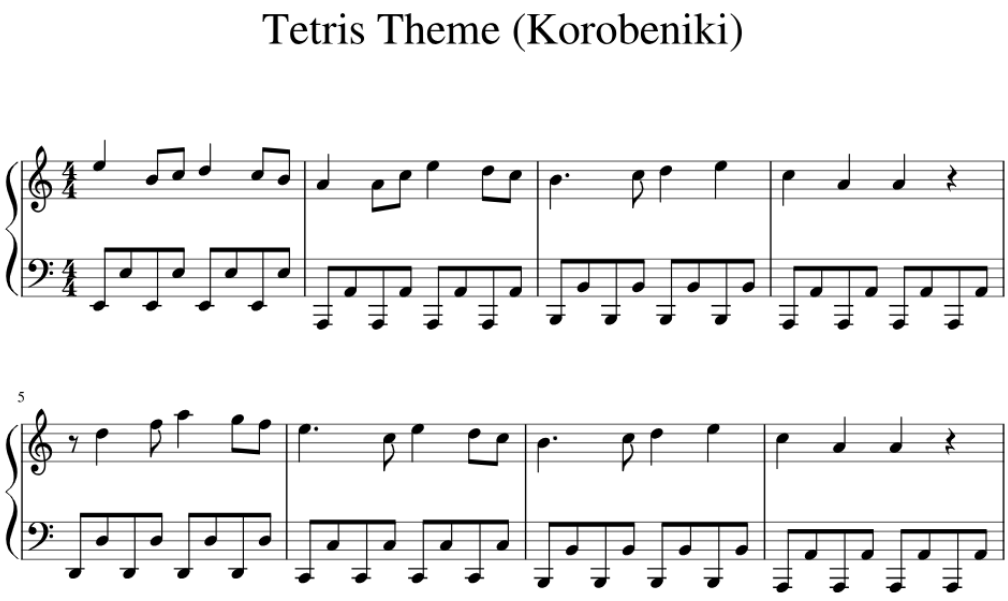

Figure 2: Korobeniki, Source: musescore.com/user/28837378/scores/5144713

Implementation:

```
play_korobeniki:
    li $v0, 31
    li $a2, 0                       # set instrument to piano
    li $a3, 100                     # set constant volume
    beq $s6, 19200, play_music_1     # first bar
    beq $s6, 18600, play_music_2
    beq $s6, 18300, play_music_3
    beq $s6, 18000, play_music_4
    beq $s6, 17400, play_music_5
    beq $s6, 17100, play_music_6
```

Figure 3: Our implementation of Korobeniki. v0 stores the MIDI Out instruction

```
play_music_1:
    li $a0, 76                  # pitch is E1
    li $a1, 600                 # duration is 600 ms
    syscall
    jr $ra

play_music_2:
    li $a0, 71
    li $a1, 300
    syscall
    jr $ra

play_music_3:
    li $a0, 72
    li $a1, 300
    syscall
    jr $ra

play_music_4:
    li $a0, 74
    li $a1, 600
    syscall
    jr $ra
```

Figure 4: A counter stored in s6 decrements every time the game loop iterates. After every half beat (since a note is played every half beat), call a function that will play the correct note (a0) for the correct duration (a1). Then return back to game loop. If multiple notes are played at once, reload a0 and a1 after the first syscall and call it again with new a0 and a1.

References (Game Over):

Figure 5: Game over sound effect, Source: musescore.com/stockphotosofrats/game-over-nsmbw

Implementation:

```
play_game_over_1:
    li $a0, 81
    syscall
    li $a0, 64
    syscall
    li $a0, 60
    syscall
    li $a0, 53
    j play_game_over

play_game_over_2:
    li $a0, 72
    syscall
    j play_game_over

play_game_over_3:
    li $a0, 75
    syscall
    li $a0, 65
    syscall
    li $a0, 62
    syscall
    li $a0, 55
    syscall
    j play_game_over
```

Figure 6: Our implementation of the music sheet (Not an exact copy). Similar to implementation for Korobeniki, but with multiple syscalls in 1 function to play chords.

3 Attribution Table

Dharma Ong (1009330109)	**Yael Lyshkow (1009560814)**
Draw playing field	Implemented I-type tetromino movement
Draw grid and initial tetromino	I-type tetromino rotations with keyboard
Collision detection (tetromino + tetromino)	Collision detection (tetromino + playing field)
Generate new tile when a tile collides	Implement functional + efficient game loop
Implemented gravity	Implemented erase row feature when a row is full
Added music (Korobeniki)	Implemented score feature
Added sound effects (game over, rotate, etc.)	Show final score when the game ends
Added game over screen	