Aalto University

ELEC-A7151 - Object oriented programming with C++

Tile Matching 4
Project Documentation

Juho Heimonen
Kalle Alaluusua
Rustam Latypov
Visa Lintunen

January 4, 2019

1 Overview

This tile-matching project is a classic Tetris with a possibility to play either Tetris or Pentis. Tetris is a well known tile-matching puzzle video game where the players objective is to guide pieces falling from the upper edge of the screen and try to keep the construction as low as possible. When a row is complete with tiles, it disappears. The game becomes more difficult as it progresses, and ends when the construction reaches the top of the screen. The pieces follow arrow-keys and can be dropped instantly by pressing 'space'.

The game rewards the player basic points for successfully landing a block, one point for each square the block contains. Elimination points are awarded based on the size of the elimination – the more rows eliminated simultaneously, the better. Points are also awarded for every instant drop, according to the height the tile is dropped from.

If the player reached top-10 in the leaderboard, the player has the option to submit his/hers high score and view leaderboard standings. This information is stored locally in a text file. The speed of the game increases specifically as a function of the amount of eliminations and not of the amount of eliminated rows. If the player didn't reach top-10, the current top-10 leaderboard is shown.

Pentis-mode of play introduces 1, 2, 3, and 5-tile-size pieces into the game. The drop-rate of these special blocks is also a function of the amount of eliminations. Pentis is considered more challenging mode of play. Tetris is faster that Pentis, but only has 7 different blocks. Pentis has 29 different pieces overall. The point-award system and game mechanics are identical to Tetris. However, the speed-up on Pentis is slower since it is already more challenging. A separate top-10 leaderboard is allocated for the Pentis game-mode.

2 Software structure

2.1 SFML

The Simple and Fast Multimedia Library (SFML) is used since the scope and the goals of the project do not require a more sophisticated platform. The following SFML objects are utilized: `RenderWindow`, `Event`, `RectangleShape`, `Font`, `Texture`, `Sprite`, `Text`, `Clock`, `Sound`, `SoundBuffer`. No other external libaries are used.

2.2 Screen structure

Each gameplay screen is initialized in `main.cpp` and inherits from an abstract class `cScreen` with a default destructor. The screens are `Menu, Game, Highscores` with unique public functions and private members. This is done to avoid repetition. The while-loop in `main.cpp` is responsible for jumping from screen to screen depending on user input. The `Highscores` screen is optimized to read from external resources only once and then display the appropriate highscore.

The `Game` screen is initialized with a 0 or a 1, indicating the gamemode. Depending on user input, the `Game` and `Menu` objects are overwriten with new ones.

2.3 End of the game

When the game ends, while-loop breaks in the `Run`-function and the program goes in to `endGame`-function defined in `endgame.hpp`. The function loads the high scores either from `t-highscores.txt` or `p-highscores.txt`, according to the game-mode the player is playing. While reading the file, every score-name-pair is pushed in to a list called `listofscores`. The `listofscores` is then compared to the new score that the current player achieved. If the players score is higher than the last element in `listofscores`, boolean `madetop10` is set true. If `madetop10` is false the current leaderboard is generated in to a string and shown to the player.

If `madetop10` is true, the player has the possibility to enter his/her name and save the score to the leaderboard. The input is done by pushing and popping characters to/from string called `input`. If the player doesn't enter name and presses enter, the score is saved under name *Unknown*. On the press of enter, the score and name are written to the highscore-file and the new leaderboard is shown to the player. *-symbol indicates the players new score on the leaderboard. On press of esc, the function returns a value to the `main`-loop.

2.4 Exeption handling and memory management

Each `.hpp` is wrapped in a `#ifndef#define...#endif` frame to avoid collisions via including. The main while-loop is wrapped in a `try..catch` frame to catch possible errors while opening/reading/closing external resources. Appropriate errors are thrown in the code.

The decision was made to favour static memory handling, thus little memory management was needed. The only `new` and `delete` commands are used in `main.cpp` appropriately. Since the `delete` command calls the default destructor defined in the abstact class `cScreen`, no copy constructor or copy assignment operator were created, abiding by the rule of three.

3 Instructions for building and using

3.1 Compiling the program

Following programs and libraries are required to compile the project:

- make (4.1-9.1ubuntu1)
- g++ (4:7.3.0-3ubuntu2)
- libsfml-dev (2.4.2+dfsg-4)

To compile the program, make sure you have the provided directories from the project git.

1. On terminal go to the project directory (Directory should contain directories: `doc`, `plan`, `res`, `src` and files `Makefile`, `README.md`.
2. In the project directory, use the following command: "`make`" to build the program.
3. To run the program, use the following command: "`./main`".
4. After the compiled files are no longer needed, you can delete them using command: "`make clean`".

3.2 Playing the game

3.2.1 Menu controls

- `Up-arrow` to go up in menu
- `Down-arrow` to go down in menu
- `Enter` to choose the wanted action
- `Esc` to go to previous menu

3.2.2 In-game controls

- `Right-arrow` to move the current block right
- `Left-arrow` to move the current block left
- `UP-arrow` to rotate the current block 90 degrees clockwise
- `L-Ctrl` to rotate the current block 90 degrees counterclockwise
- `DOWN-arrow` to drop the current block down one step
- `Space` to drop the current block instantly to the bottom
- `Esc` to pause the current game

3.2.3 End-game controls

- ASCII-127-characters to type name
- `Enter` to enter the name
- `Esc` to return to the main menu

3.2.4 Gameplay strategy

The player can eliminate up to five rows at the time. The points gained per elimination grow exponentially as the size of the elimination increases, which encourages player to focus on larger eliminations. In addition, both Tetris and Pentis game modes increase in difficulty as the number of eliminations increases, which incites the player to avoid small eliminations.

Traditionally, the falling speed of the tetrominoes increases as the number of eliminations increases. This holds true for our implemention. However, this is not the case in Pentis game mode. As 5-tile blocks, pentominoes, are significantly more difficult to place on the stack than smaller blocks, these appear more frequently as the number of eliminations increases. To be precise, the initial probability of getting a pentomino is 0. After every fifth elimination, this probability increases five percent, until after 100 eliminations player is given pentominoes only. The smaller blocks, monominoes, dominoes, trominoes and tetrominoes are distributed such that the player is 2^n times as likely to get a block of size n than a monomino.

The blocks behave according to the Super Rotation System (SRS), the current Tetris Guideline standard for how tetrominoes rotate and what wall kicks they may perform. This allows for spin moves that let the player fix some otherwise difficult situations. The behaviour of dominoes, trominoes and pentominoes is derived from the tetromino behaviour.

4 Testing

4.1 Gameplay and movement

Tetramino gameplay was tested exhaustively to verify that the game follows SRS. The SRS specific spins listed in `http://tetris.wikia.com/wiki/List_of_twists` were performed on a simulated stack with falling due to gravity disabled. A pass/fail grading system was used and the game passed the tests.

The block distribution and the increment in difficulty level were tested and approved by the crew and a non-affiliated third party member. These test were purely qualitative as we do not have the means to quantify fun.

4.2 End of the game

When the player reaches top-10 on the leaderboard, he/she has the possibility to enter his/hers name. This input field accepts 9 ASCII-127 characters, excluding enter, space and tab. According to our manual fuzz testing, the player should not be able to break the leaderboard with unexpected input.

4.3 Memory leaks

For memory safety a stack tracing tool Valgrind was used. The game does not free all resources at termination because SFML uses its own memory pools. This is seen when terminating the game with valgrind as "Possibly lost" section does not indicate zero. The amount of memory claimed lost by Valgrind does not increase with the lifetime of the game or the number of games but stays constant.

5 Work log

Week 1 (12.11.-18.11.)

- As a group
  - Deciding the main functionalities the game will have
  - Designing the rough structure of the game
- Juho Heimonen
  - Getting familiar with SFML
  - Creating the window and tile

    Work hours: ∼ 3
- Kalle Alaluusua
  - Designing initial tetromino representation and movement

    Work hours: ∼ 3-4
- Rustam Latypov
  - Learning SFML
  - Setting up OS X environment

    Work hours: ∼ 3-4
- Visa Lintunen
  - Learning how to use SFML in Visual Studio
  - Learning how to use SFML

    Work hours: ∼ 5-6

Week 2 (19.11.-25.11.)

- As a group
  - Final decisions on controls, mechanics, object interaction
  - Learning the basics of SFML

- Juho Heimonen
  - Implementing gravity and simple player movement
  - Simple sidebar for points
  - first spawning pieces
  - first piece freezing functionality

    Work hours: ∼ 8
- Kalle Alaluusua
  - Implementing initial class representation for tetrominoes
  - Implementing primitive tetromino rotation
  - Getting familiar with drawing tetrominoes using SFML

    Work hours: ∼ 8-10
- Rustam Latypov
  - Designing the SFML-based screen architecture and UI layout

    Work hours: ∼ 8-10
- Visa Lintunen
  - Implementing simple SFML-based window with moving tile as a basis of the project
  - Learning how to write a makefile
  - Learning how to include external libraries in linux environment

    Work hours: ∼ 8

Week 3 (26.11.-02.12.)

- As a group
  - First playable game out
- Juho Heimonen
  - Custom consistent cross-platform controls for player movement
  - Speed up in tetris
  - Remove animations and final tile graphics
  - final end game logic and consistent spawning of pieces
  - Correct freezing of pieces
  - architectural improvements
  - removing full rows

    Work hours: ∼ 18

- Kalle Alaluusua
  - Researching and implementing advanced tetromino rotation system including wall kicks
  - Implementing mono-, do-, tro- and pentominoes
  - Representing the blocks in a map holding the corresponding colors and using character representation as keys

    Work hours: ∼ 14-16
- Rustam Latypov
  - Merging the implementations of other team members
  - Architecture implemented with one game mode and single highscore

    Work hours: ∼ 16-18
- Visa Lintunen
  - Writing first makefile
  - Implementing end of the game
  - Implementing writing and reading the high score-files

    Work hours: ∼ 14-16

Week 4 (03.12.-09.12.)

- As a group
  - Final decisions on scoring logic and visuals
  - Debugging in linux-environment
- Juho Heimonen
  - Sounds
  - Memory testing
  - Memory debugging
  - Gameplay tweaks
  - Documentation

    Work hours: ∼ 15
- Kalle Alaluusua
  - Designing and implementing advanced block randomization including permutating tetrominoes and increasing pentomino propability

    Work hours: ∼ 14-16
- Rustam Latypov
  - Final architecture structure with both gamemodes

  - Debugging architecture
  - Fixing memory leaks
  - Researching UX and developing the endgame accordingly

    Work hours: ∼ 10-12

- Visa Lintunen
  - Writing the final makefile
  - Fixing bugs in visuals
  - Improving visuals
  - Creating the final background for the game

    Work hours: ∼ 10-12

Week 5 (10.12.-14.12.)

- As a group
  - Final version ready
- Juho Heimonen
  - Documentation
  - Gameplay testing
  - Tweaking gameplay

    Work hours: ∼ 2

- Kalle Alaluusua
  - Testing game dynamics and altering block distributions accordingly
  - Testing block mechanics for bugs
  - Writing documentation

    Work hours: ∼ 8-10

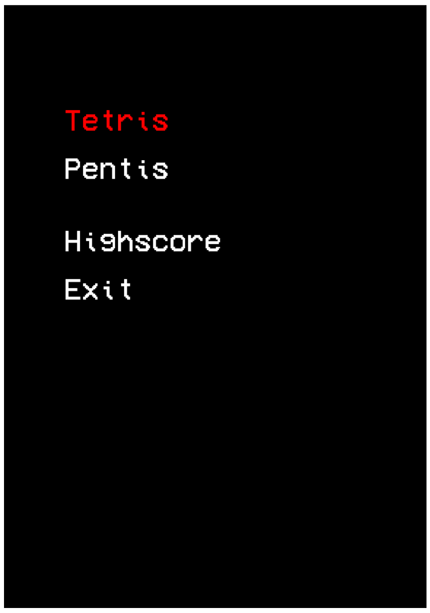
- Rustam Latypov
  - General gameplay testing and minor impovements
  - Documentation and code cleanup

    Work hours: ∼ 6-8

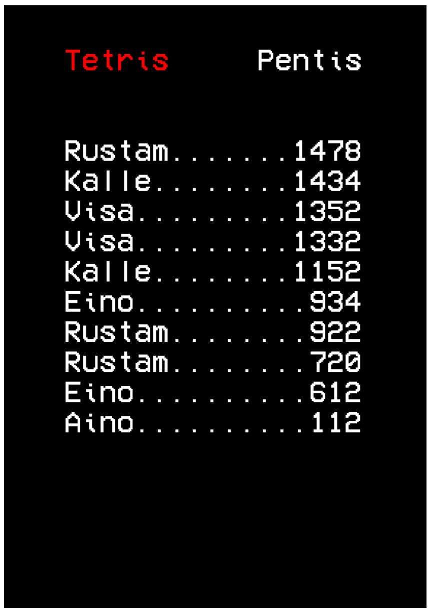
- Visa Lintunen
  - Writing the documentation
  - Trying to implement automatic scaling for the game window with multi-platform compability

    Work hours: ∼ 10-12

6 In-game pictures



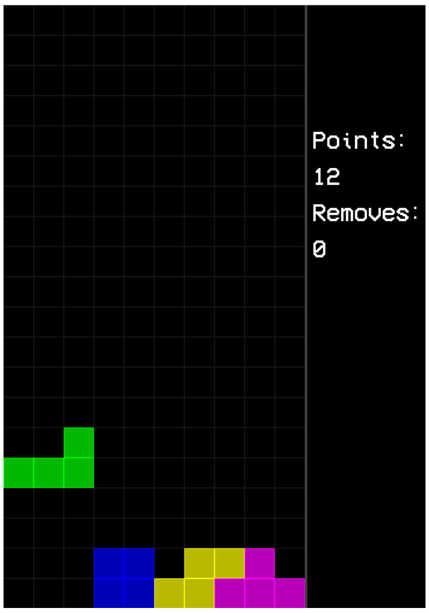
(a) Main menu



Tetris Pentis

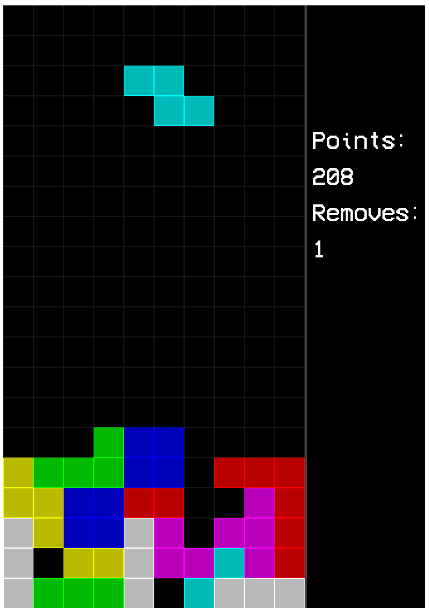
```
Rustam.......1478
Kalle........1434
Visa.........1352
Visa.........1332
Kalle........1152
Eino..........934
Rustam........922
Rustam........720
Eino..........612
Aino..........112
```

(b) Highscores



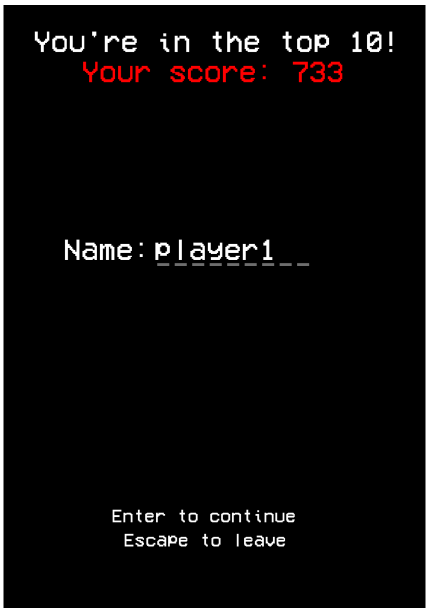
```
Points:
12
Removes:
0
```

(a) Tetris



```
Points:
208
Removes:
1
```

(b) Pentis

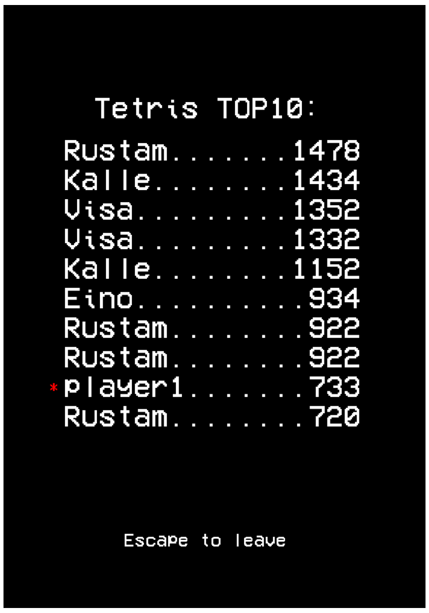


```
You're in the top 10!
Your score: 733

Name: player1

Enter to continue
Escape to leave
```

(a) End of the game name input



```
Tetris TOP10:
Rustam.......1478
Kalle........1434
Visa.........1352
Visa.........1332
Kalle........1152
Eino..........934
Rustam........922
Rustam........922
*player1.......733
Rustam........720

Escape to leave
```

(b) End of the game highscores