Aalto University

ELEC-A7151 - Object oriented programming with C++

Tile Matching 4

Project Documentation

Juho Heimonen
Kalle Alaluusua
Rustam Latypov
Visa Lintunen

January 7, 2019

1 Overview

This tile-matching project is a classic Tetris with the possibility to play either Tetris or Pentis. Tetris is a well known tile-matching puzzle video game where the players objective is to guide 4-tile-size blocks falling from the upper edge of the screen and try to keep the stack as low as possible. Pentis introduces 1-, 2-, 3-, and 5-tile-size blocks into the game and its difficulty increases differently than in Tetris.

The blocks are controlled by arrow-keys and can be dropped instantly by pressing `space`. When a row is complete with tiles, it disappears. The game ends when the stack reaches the point where the new spawning block overlaps with the stack.

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

The leaderboard keeps track of the top-10 scores and the names of those players. This information is stored locally in a text file. If the player reaches the top-10, the player has the option to submit his/hers name and score and view leaderboard standings. If the player doesn't reach the top-10, the current leaderboard is shown. Separate leaderboards are kept for the game modes.

The difficulty of both game modes increase specifically as a function of the number of eliminations and not of the number of eliminated rows. Tetris spawns the same distribution of blocks but the falling speed increases. The falling speed in Pentis stays constant but the 5-tile-size blocks starts to spawn more frequently. Tetris is faster that Pentis, but only has 7 different blocks. Pentis has 29 different blocks overall and is considered a more challenging mode of play.

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 approach. 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, according to the gamemode. When a game ends or is exited from, the `Game` and `Menu` objects are respectively overwriten with new ones.

2.3 End of the game

When the game ends, the while-loop breaks in the `Run`-function of the `Game`-object and the program calls the `endGame`-function defined in `endgame.hpp`.

If the player makes the top-10, the player is asked to enter his/her name to save the score to the leaderboard. If the player doesn't enter a name, the score is saved under *Unknown*. The *-symbol points to the players new score on the leaderboard. The player can then exit the screen via `esc` and the function returns to the while-loop in `main.cpp`.

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)

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 a time. The points gained per elimination grow exponentially as the size of the elimination increases, which encourages player to favor larger eliminations. In addition, both Tetris and Pentis game modes increase in difficulty as the number of eliminations increases, which further 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 the implementation of Tetris. However, this is not the case for the Pentis game mode. Since 5-tile-size 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 to favour larger pieces by a factor depending on the block size n and the collection size. A collection contains all the unique pieces of size n. A player is $2^n * \textit{collection size}$ time as likely to get a block of size n than a monomino. The

game employs true randomness with `srand((time(NULL)))` that uses current time as the random seed.

The blocks are controlled 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 give the player more control. The behaviour of dominoes, trominoes and pentominoes is derived from the tetromino guidelines.

4 Testing

4.1 Gameplay and movement

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. The game passed the tests with all blocks.

The block distribution and the increments in difficulty were tested and approved by all developers 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 manual testing, it should not be possible 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 due to the memory pools of SFML. This is seen shown by Valgrind as a "Possibly lost" section. The amount of memory claimed lost by Valgrind does not increase with the lifetime of the game or the number of games, thus pointing to constant memory usage of SMLF. Since the "memory leak" is constant, it is not an issue.

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: $\sim$ 3-4
- Rustam Latypov
  - Learning SFML
  - Setting up OS X environment

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

    Work hours: $\sim$ 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 blocks
  - first block freezing functionality

    Work hours: $\sim$ 8

- Kalle Alaluusua
  - Implementing initial class representation for tetrominoes
  - Implementing primitive tetromino rotation
  - Getting familiar with drawing tetrominoes using SFML

    Work hours: $\sim$ 8-10

- Rustam Latypov
  - Designing the SFML-based screen architecture and UI layout

    Work hours: $\sim$ 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: $\sim$ 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 blocks
  - Correct freezing of blocks
  - architectural improvements
  - removing full rows

    Work hours: $\sim$ 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: $\sim$ 14-16

- Rustam Latypov
  - Merging the implementations of other team members
  - Architecture implemented with one game mode and single highscore

    Work hours: $\sim$ 16-18

- Visa Lintunen
  - Writing first makefile
  - Implementing end of the game
  - Implementing writing and reading of the highscore-files

    Work hours: $\sim$ 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: $\sim$ 15

- Kalle Alaluusua
  - Designing and implementing advanced block randomization including permutating tetrominoes and increasing pentomino propability

    Work hours: $\sim$ 14-16

- Rustam Latypov
  - Final architecture structure with both game modes
  - Debugging architecture
  - Fixing memory leaks
  - Researching UX and developing the endgame accordingly

    Work hours: $\sim$ 10-12

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

    Work hours: $\sim$ 10-12

Week 5 (10.12.-14.12.)

- As a group
  - Final version ready
- Juho Heimonen

  - Documentation
  - Gameplay testing
  - Tweaking gameplay

    Work hours: $\sim$ 2

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

    Work hours: $\sim$ 8-10

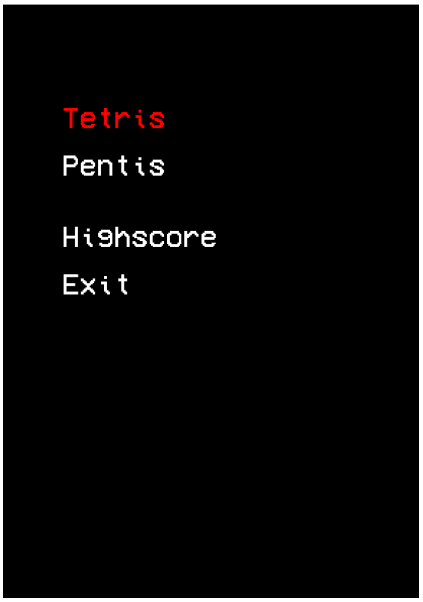
- Rustam Latypov
  - General gameplay testing and minor improvements
  - Documentation and code cleanup

    Work hours: $\sim$ 6-8

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

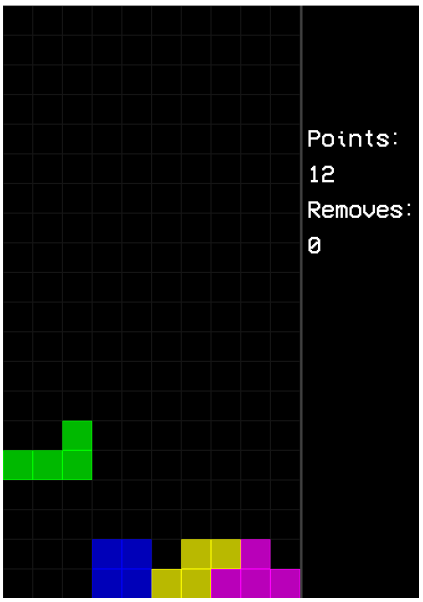
    Work hours: $\sim$ 10-12

6 In-game pictures

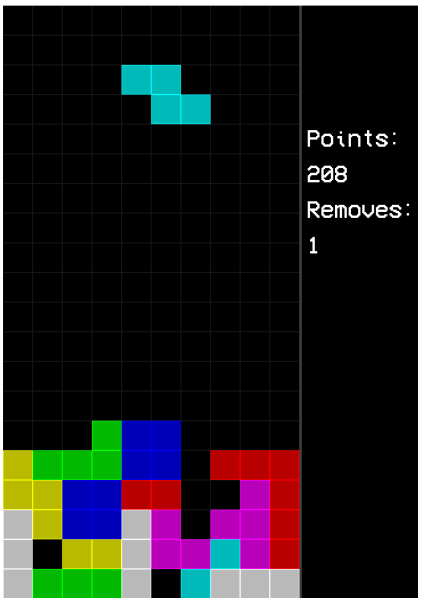


(a) Main menu

(b) Leaderboard

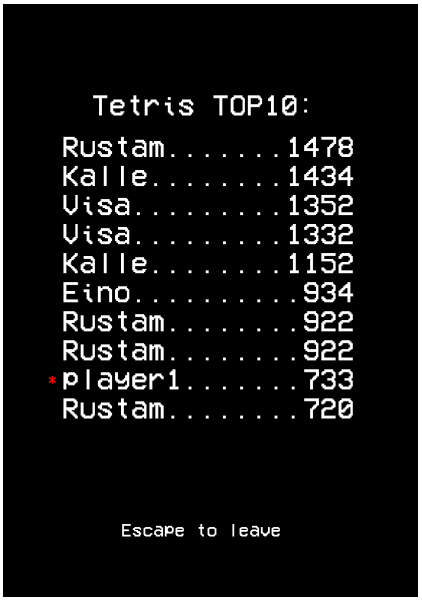


(a) Tetris



(b) Pentis

(a) End of the game name input



(b) End of the game leaderboard