Brick game: Tetris. Documentation

SIONAPAE

05.10.24

1 Project Description

Implementation of the "Tetris" game in C using a structured approach.

2 Requirements

To run the program, you need:

- GCC compiler or any other C++-compatible compiler
- GNU Make to build the program
- The libraries check.h and ncurses.h

3 How to Build and Run the Game

1. Run the command 'make all' to build the program.
2. Start the game by running 'make run'.
    - all
    - install — compiles the Tetris source file
    - run — runs the game
    - uninstall — deletes the Tetris executable
    - clean — removes build files
    - dvi — project documentation
    - dist — archives the project
    - test — runs unit tests
    - gcov_report — generates an HTML coverage report

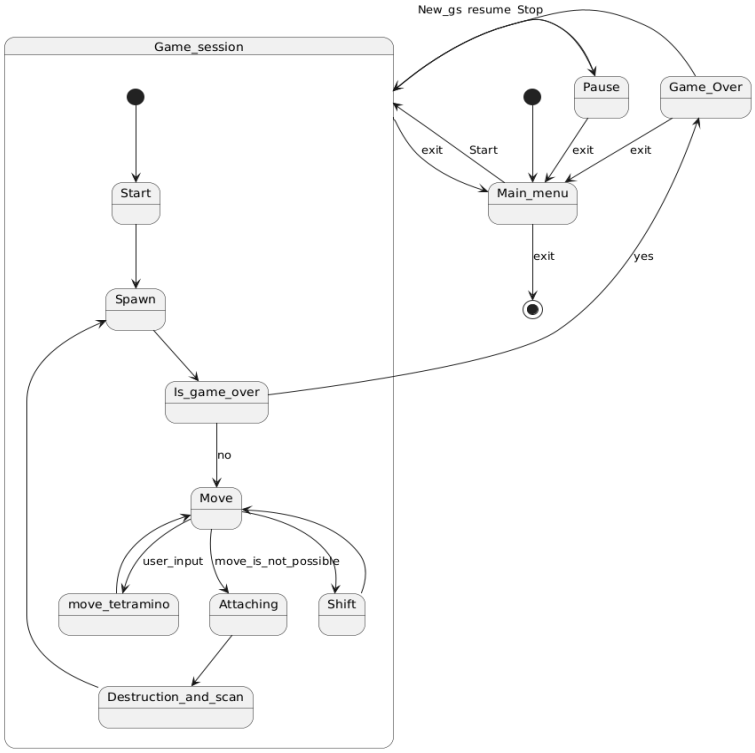
4 Controls

The player uses keyboard keys that simulate physical buttons on a real console to control falling blocks:

- D key — Left arrow — move the block left.
- A key — Right arrow — move the block right.
- F key — Down arrow — make the block fall.
- S key — accelerates the fall of the block.
- W key — Up arrow — rotate the block.
- P/SPACE key — pause the game.
- N/ENTER key — start a new game or continue.
- q/ESCAPE key — quit the current game, exit the application.

5 Program Architecture

A finite state machine for a specific implementation of the Tetris game.



The program is built around a Finite State Machine (FSM) that manages the game's logic. The `main()` function sets up the environment, starts the game loop via `game_loop()`, and handles game termination.

The entire code structure is organized as follows:

5.1 `main()`: Entry Point

This function initializes the game environment, runs the main game loop, and cleans up when the game finishes.

- **Random number generator initialization:**

```
srand(time(0));
get_random();
```

  Here, the random number generator is initialized to generate random Tetrimino pieces.

- **Game objects initialization:**

```
Game_Objects_t* params = get_instanse();
*params = init_empty_game_objects();
```

  The `params` object holds information about the current game state, user actions, and game objects.

- **Ncurses initialization and game loop start:**

```
#ifndef debug_bro
init_bro_ncurses(&params->views);
game_loop();
terminate_ncurses_bro(&params->views);
#else
```

```
game_loop();
#endif
```

Depending on the compilation mode, the game either uses ncurses for a console user interface (CUI) or runs in debug mode without ncurses.

5.2 `game_loop()`: The Main Game Loop

The `game_loop()` function runs until the game is over. It processes the main game states and user actions.

- **Game state initialization:**

```
State prev = START;
```

The variable `prev` stores the previous state of the game to return to it after pause or other states.

- **Main game loop:**

```
while (params->game_is_running) {
    draw_static(params);
    main_game_fsm(params);
    game_session(params, &prev);
}
```

The main loop continues as long as `game_is_running == true`. It processes game states and user actions within the loop.

5.3 `game_session()`: Managing the Game Session

The `game_session()` function controls the actual gameplay (such as moving the Tetriminos, counting time, and handling pauses).

- **Game session initialization:**

```
if (params->state == START) {
    session_is_running = true;
    params->state = *prev;
}
```

If the game begins in the `START` state, the previous game state is restored, and a new game session starts.

- **Game session loop:**

```
while (params->state != PAUSE && params->state != GAME_OVER && params->state != MAIN_MENU)
    fsm_game_session(params);
    key = getch();
    userInput(getSignal(key), session_is_running);
    countTime(params);
}
```

The loop continues until the game ends (GAME_OVER) or is paused. Inside the loop, user actions (move, pause, quit) are processed, and the game state is updated.

- **Saving high scores:**

```
if (params->gameInfo.score > params->gameInfo.high_score)
    write_high_score(params->gameInfo.score);
```

If the player achieves a new high score, it is saved.

6 Technologies and Libraries Used

- **Compiler version:** Recommended version — GCC 9.3
- **External libraries:**
  - `ncurses.h` — library for working with console user interfaces (CUI). It provides functions for terminal control, input processing, and window management in text mode.