CSE 3150 – Lab 4
Building the 2048 Game in C++

By: Dr. Justin Furuness

Overview

In this lab, you will implement a simplified version of the game **2048**. You will start from provided starter code and incrementally add functionality until your program passes the given test suite.

By the end, you will:

- Understand how to manipulate 2D arrays in C++.
- Use Standard Template Library (**STL**) tools such as `std::vector`, `std::stack`, and `std::algorithm`.
- Apply iterators, the `auto` keyword, and adapters effectively.
- Implement game mechanics such as shifting, merging, and undo.
- Verify your program using automated tests.

Your work must be committed to a file called **solution.cpp** in your GitHub repository **cse3150_lab_4**. Your submission is complete when all tests pass.

Game Rules Recap

The game is played on a 4 × 4 board:

- Each turn, the player chooses a direction: left (`a`), right (`d`), up (`w`), or down (`s`).

- Tiles slide in that direction. Adjacent tiles of the same value **merge** into one tile of double the value.
- After each move, a new tile (usually a 2) spawns in an empty cell.
- The player can type `u` to undo the last move.
- The player can type `q` to quit.

Starter Files

You are given two files:

- `starter.cpp` — Skeleton code for the game (already configured to write board state to CSV).
- `test_game.py` — Pytest tests that validate your implementation.

You will build your solution by editing and renaming `starter.cpp` into **solution.cpp**.

Step 1: Representing the Board

The board is a 4 × 4 grid of integers. In C++, this can be stored as:

```
std::vector<std::vector<int>> board(4, std::vector<int
    >(4, 0));
```

Key C++ tools for this step:

- Use `std::vector` instead of raw arrays for flexibility.
- Access rows with `board[i]` and individual cells with `board[i][j]`.
- Use range-based `for` loops with `auto&` to traverse rows and cells.

Step 2: Shifting and Compressing Tiles

To shift tiles left, right, up, or down, we first need to *compress* them by removing zeros. A recommended STL approach is to use the copy_if function. Start with an empty vector compressed, and then using copy_if and a back_inserter, you can copy the row to your compressed vector. copy_if also comes with the ability to filter as you copy, read the documentation for how to do this and pass in a lambda to filter out the zeros.

```
std::vector<int> compressed;
```

This uses:

- `std::copy_if` from `<algorithm>` to filter tiles.
- A `std::back_inserter` from `<iterator>` to append efficiently.
- A lambda expression to keep only non-zero entries.

After compression, pad with zeros to return the row to length 4.

Step 3: Merging Tiles

When two adjacent tiles are equal, merge them:

Then call the compress step again to remove the zeros created by merging. This ensures the ”no double merge” rule is respected.

Step 4: Spawning New Tiles

After each valid move:

- Collect empty positions in a vector of pairs.
- Use `std::mt19937` and `std::uniform_int_distribution` from `<random>` to select a random empty cell.
- Assign a 2 (90% of the time) or 4 (10%).
- If this syntax is confusing, use chatGPT to help since we haven’t covered this in class. AI is a great way to look up syntax, and you can choose the ”instant” model to get a faster answer since this is a simple problem with AI.

Step 5: Implementing Undo

To support undo:

- Use the adapter `std::stack` from `<stack>` to store previous board states.
- Before applying a move, push a copy of the board.
- On undo (`u`), restore from the stack if not empty.

Step 6: Implementing Score

To support undo:

- Modify the stub for the score function to take instead a template for a reference class Board and make computing the score more abstract.
- Calculate the score properly.

Step 7: Putting It All Together

Your `main()` loop should:

1. Print the current board and (already handled in starter code) write it to CSV.
2. Read a character command from input.
3. Use a `switch` statement for actions (`a`, `d`, `w`, `s`, `u`, `q`).
4. For moves: push the board to history, compress, merge, compress again, then spawn a new tile.
5. For undo: restore from the stack.
6. For quit: break the loop.

Testing Your Code

Run the provided tests with:

```
pytest test_game.py
```

Your code is correct when all tests pass. Push your final version to GitHub as **solution.cpp** under the repo cse3150_lab_4