Tetris Requirements Specification

Sep.15th, 2021 | CSCI-GA-2440-001 Software Engineering

1. Project Team

Ken S. Zhang | sz1851@nyu.edu

Qingyang Li | ql2048@nyu.edu

2. Introduction

The goal of this course project is to build a Python-based **Tetris** game. This game provides features that follow the widely-accepted rules. This document describes the requirements of this program.

3. Use Cases

UC1 Set Up Game

UC2 View Leaderboard

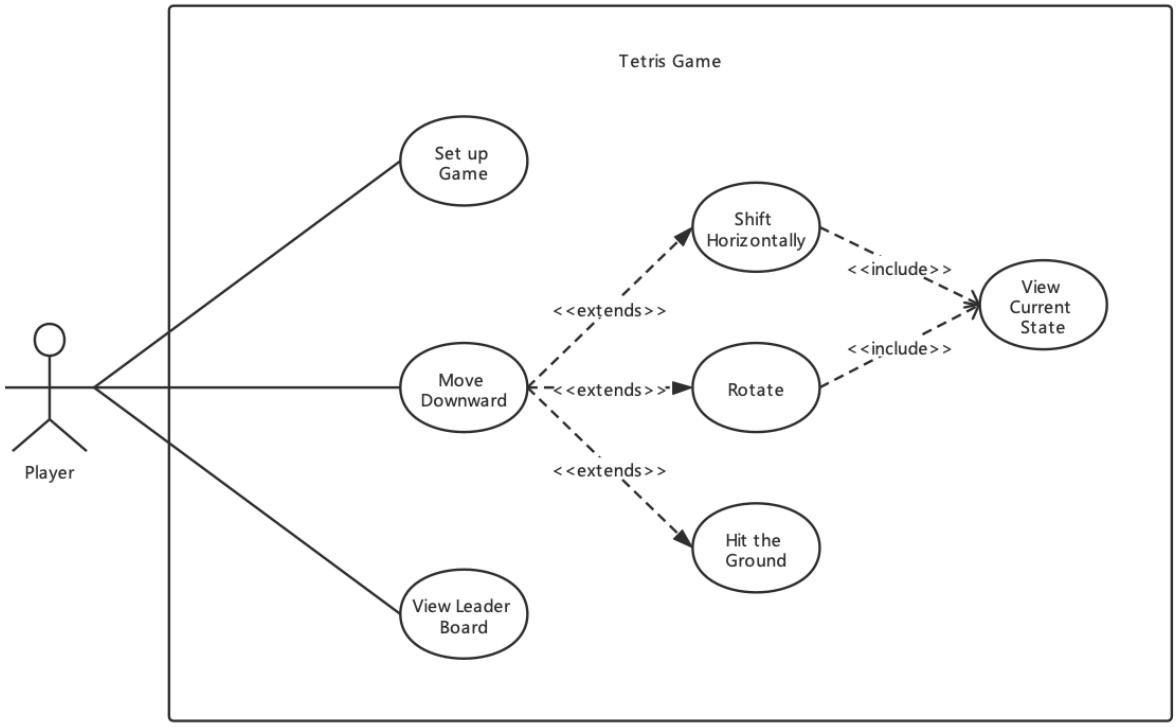
UC3 Move Downward

UC4 Shift Horizontally

UC5 Rotate

UC6 View Next Tetromino

UC7 Hit the Ground



UC1 Set Up Game

1.1 Preconditions:

None.

1.2 Main Flow:

Right after the game get started, the Game Setup dialog will show to prompt the player enter the level of difficulty [E1] (clock speed, shape complexity of Tetrimino, etc.) for this game and the name of the player [E2] [E3].

1.3 Sub Flows:

None.

1.4 Alternative Flows：

[E1] The level of difficulty is measured by integers ranging from 1 to 5. If the input number doesn't fall into this range or have type error, the game asks the player to retype again.

[E2] The name can't be empty. If a player enters an empty string, the game asks the player to retype again.

[E3] If the Quit button is pressed, the whole program will close.

UC2 View Leaderboard

2.1 Preconditions:

The player has failed and is out of the game.

2.2 Main Flow:

Right after the player finishes the game, the View Leaderboard dialog will show to prompt the player view his/her score and the historical top-10 scores for this game [E1].

2.3 Sub Flows:

None.

2.4 Alternative Flows：

[E1] If the Quit button is pressed, the whole program will close.

UC3 Move Downward

3.1 Preconditions:

The player has successfully set up the game and entered the main game interface.

3.2 Main Flow:

The game space is basically a rectangle with internal grids. There is a Tetrimino on the fly. The game is clock-based. Every time the clock ticks, the Tetrimino will move one step downward no matter what. The player has no control of this action. The player can perform [S1] between two consecutive clock ticks. This action will keep going on unless [S2].

3.3 Sub Flows:

[S1] The player can either shift the Tetrimino horizontally [UC4] or rotate the Tetrimino [UC5] by referring to the current game space and [UC6].

[S2] After the Tetrimino move one step downward, based on the position of the Tetrimino, the game state may change to [UC7].

3.4 Alternative Flows：

None.

UC4 Shift Horizontally

4.1 Preconditions:

The current Tetromino is still moving downward.

4.2 Main Flow:

The player may move the current Tetromino left or right by one grid [S1] [E1]. After that, the horizontal location of the current Tetromino is altered, and the Tetromino continues to move downward [UC3].

4.3 Sub Flows:

[S1] The player clicks the Move Left or Move Right button. If the movement is valid, the Tetromino will move to the corresponding location accordingly.

4.4 Alternative Flows:

[E1] If the current Tetromino has reached the leftmost/rightmost column or shifting will conflict with non-empty cells, Move Left/Right will not succeed, and nothing will happen.

UC5 Rotate

5.1 Preconditions:

The current Tetromino is still moving downward.

5.2 Main Flow:

The player may rotate the current Tetromino clockwise or counter-clockwise by 90 degrees [S1] [E1]. After that, the spatial position of the current Tetromino is altered, and the Tetromino continues to move downward [UC3].

5.3 Sub Flows:

[S1] The player clicks the Rotate Clockwise or Rotate Counter-clockwise button. If the movement is valid, the Tetromino will rotate its spatial position accordingly.

5.4 Alternative Flows:

[E1] If the current Tetromino is "out of boundary" after rotation or rotation will conflict with the non-empty cells, Rotate Clockwise/Counter-clockwise will not succeed, and nothing will happen.

UC6 View Current State

6.1 Preconditions:

The current Tetromino is moving downward.

6.2 Main Flow:

While the current Tetromino is moving downward [UC3], the player may check current state of the game (e.g. current timestamp, shape of next tetromino [S1]) and decide if further actions [UC4, UC5] need to be taken.

6.3 Sub Flows:

[S1] If this is not done, the system will generate the next Tetromino randomly.

6.4 Alternative Flows:

None.

UC7 Hit the Ground

7.1 Preconditions:

The Tetrimino has moved one step downward.

7.2 Main Flow:

If the bottom boundary of the Tetrimino touches any non-empty cells, this Tetrimino is said to hit the ground. Then the program will detect if there is any complete row (i.e. row with no empty cells). Rows that satisfy this criterion will be erased all at once. Thus, the whole game space will be shifted downward and filled with empty rows to compensate for loss of rows. [E1] Finally, a new Tetrimino will appear at the top of the game space.

7.3 Sub Flows:

None.

7.4 Alternative Flows：

[E1] If there is still any non-empty cell in the top row, the player fails the game. Subsequently, the player's score will be recorded and [UC2] will show up.

4. Nonfunctional Requirements

NR1. Performance

The program should accept the player's inputs during the game, but execute only the valid operations given a player's input to the system. All functions shall be completed quickly.

NR1.1 User Response

The program shall respond to any user input within 0.01 seconds.

NR2. Usability

A player should be able to determine quickly what type of operations they can/have to perform.

NR2.1 Player options

A player shall only have access to functionality that is allowed at a given time.

NR2.2 User Interface

The program shall allow a player to interface with it through keyboard and mouse. During the game, all operations can be achieved using 4 Arrow buttons of the keyboard. The amount of mouse input shall be minimized by the system to include only entering level of difficulty and player name.

NR2.3. User Errors

The program shall catch invalid input from all text fields in the system.

5. Constraints

The program should be written in Python3.

6. Development and Target Platforms

- Windows 10 & MacOS
- Intel x86 Micro-processors
- Visual Studio Code