CS246 Spring 2021 Project - Biquadris - Demo

By: j538xu, pxlin, y82hou

Introduction

In this demo, we will run through the program and show all features. Each test with different purposes has its own .in .out and .args files. We have included all tests' args (if they have it) and in files in the zip.

Test Files and Descriptions

T1 shows effects of movement commands: left, right, clockwise, counterclockwise, drop; show input command checking system (include invalid inputs), and also how upper and lower case can both be recognized.

T1: no args file

T1.in: Left right right right right right right right clockwise right right 10 left I dr le lef right counterclockwise right right rIghT RIGHt ri ri ri 10ri 10lef clock count 2clock 2count count c count cou co cl clock 14clock clockwise clo count clo count clo clo cou cou cou clo clo lef le cout rig ClO dr I cou 10ri clo dro S count clock count 5cou 5clo Z 5cou 6count 7cloc I ri 9count 10 clo l 4count 3clo 17count J 10count 10clo 10count T 3ri 14clo 6lef 7count O 3count 8clock Q 3count B count j t

T1.in first shows how moving the brick to a part not in the board will not be moved. Then moves the brick for both players to the right most then back to the left most. Then for player 1, change the type to I and drop, and drop right away for player 2. Then let player 1 have an I brick, change it to vertical, move it to the rightmost, and try to change it to horizontal, it will not turn since we always keep the bottom left corner at the same position before and after turning it. During that process, we also change the bricks to clockwise and counterclockwise multiple times to test those two commands. Then it changes player 2's brick to any possible type of brick, and rotate them to show that their rotations are correct. This also checks that the bricks all appear at the correct position (i.e., there are three lines above the top row of the brick). Notice that for each command, prefix allows the command to be used in a loop, writing incorrect input will not cause the program any trouble, an error display will be displayed and the next input will be taken. If the user enters commands that are close enough to a known command, such as ri and lef, it will be recognized and used as a command.

T2 shows effects of text-only mode (-text) and hint

T2.args: -tex -text -texttttt

T2.in: I dr dr I 4ri dr dr I clo 8ri dr dr I clo 9ri dr dr I hint clo 10ri dr dr I clo hint

T2 shows the -text command line feature, as it disables the graphics display. It also shows how incorrect command line arguments will be ignored. Then, the command line fills player 1 with I blocks until a single brick can clear the bottom row. Then player 1 will ask for a hint where the program will run simulations on the backend to find the most lowest drop for the next brick, in which is used after calling the hint and the last row is cleared. Then player 1 will rotate the I block and ask for a hint, this hint demonstrates how the program can recognize that the block being drop needs to be rotated before being dropped to reach the maximum amount of blocks at the lowest possible height. One drop is called per player2's turn to skip back to player 1. Each brick upon appears also shows how the sequence is following the sequence files.

T3 shows effects of reading files using command line argument and norandom, sequence and use command random to generate new blocks with given probability.

seq1.txt : J L

seq2.txt : Z T

seq3.txt : I O S S O S J J

seq4.txt: S S J S Z T Z Z L I

cmd.txt: random drop drop drop drop dro drop norandom seq4.txt drop drop drop drop

T3.args -scriptfile1 seq1.txt -scriptfile2 seq2.txt

T3.in: clock drop drop drop drop drop 3levelup noran seq4.txt drop 3levelup noran seq3.txt drop drop drop drop drop seq cmd.txt

T3 shows the command line arguments -scriptfile1 and scriptfile2, which the first few drops shows how level 0 no longer follows the default sequence files and reads the types in new files that are imported by those commands. They also show that when it reaches the end of the file, it starts to read the start of the file again. Then we level up the second player into third level and we can use the norandom function to stop generating types in given probability. Instead, it generates types from seq3.txt. And we level up 3 times for the first player as well and in the next few drops, we will see how the types of blocks are generated by the new files we use. Now, we use the sequence function to read commands from the file cmd.txt. Now we use the random command to set player two back to generate types by given probabilities. And in the next few drops we can see it is set to random generated mode. After that we show that we can also use norandom command in the command.txt and generate types from the given file. We can check that in the next few drops.

T4 shows command line startlevel and other sorts of level requirements.

T4.args -startlevel 1

T4.in dr dr dr levelu 10ri dr 2levelu 100ri dr 1000ri dr ri ri 2ri dr 10levelu ri 2ri 10 ri ri 5dr leveld ri ri 100leveld

T4 shows the command line startlevel, the first few drops shows how level 1 no longer follows the sequence files and randoms the types. Then 2levelu shows how level up works with a multiplier and 100ri and 1000ri shows how rights stop at limit, and only applies heavy once. Same goes to ri ri 2ri, this shows how heavy applies once to each. Then 10levelu shows how levelup only goes to 4, and level 4 also have heavy blocks that works the same as level 3, heavy is still applied when the user calls a movement that cannot be made, and 5 drops without a clear adds a type * in the middle. The graphics will show different colour for each type, and brown for type *.

T5 shows each of the special actions, restart, interactions between special actions and level, how heavy forces a turn but not the command down. Score is also shown.

T5.args: nothing

T5.in: I dr I dr I dr I dr I 4ri dr I 4ri dr I 4ri dr I 4ri dr I clo 8ri dr I clo 8ri dr I clo 9ri dr I clo 9ri dr I clo 10ri dr blind I clo 10ri dr heavy I 4ri dr I dr I dr force p p m I 4ri dr dr restart I dr I dr I dr I dr I 4ri dr I 4ri dr I 4ri dr I 4ri dr I clo 8ri dr I clo 8ri dr I clo 9ri dr I clo 9ri dr I clo 10ri dr heavy 4levelu lef ri 2ri 2ri 20 down lef 2ri dr dr

T5 shows each special action on board and display. First the blind feature shows how some cells in the board are turned to ? and yellow blocks on display. Then heavy shows how blocks are now heavy when moving it. Then force is shown by forcing the next type, inputting the wrong type does not make the program crash. Then the game is restarted, the same commands are run until another special action. Then heavy with 4levelu is shown how 2 heavy is applied to the player. Then it shows how when heavy forces a block to the bottom, it automatically switches players, then down shows how when it reaches bottom by down, it does not switch players until a drop is called.

T6 shows what happens when the middle of a brick is cleared.

T6.args :nothing

T6.in: l 2clo dro dro 2clo 3ri dro dro 2clo 6ri dro dro i clo 10ri dro dro l count 8ri dro RESTArt l 2clo dro dro 2clo 3ri dro dro 2clo 6ri dro dro s clo 10ri dro

T6 shows how when the middle of a brick is cleared, the top half also falls down. This test has two parts, separated by restart. First part shows how I and L's middle is cleared, the second part shows how S's middle is cleared. That is, when height of S is 3, and the middle is cleared, the top block will drop to the bottom row.

T7&8 shows the basics of our bonus features. Mainly how we can allow different amount of player, different height and width, and colour-blind mode.

T7.args -enablebonus

T7.in 1 200 200 y Left right dr right 5right rig ock count 2clock 2count count c count cou co cl clock dr 14clock clockwise clo dr count clo count clo clo dr cou cou cou clo clo lef le cou 8clock Q 3count B count j t

T7 shows our bonus with some random commands. In this case, we set player count to 1, and colour blind mode to make all bricks one colour. We also allowed customized height and width, in which we entered as 200 but capped to max(20).

T8.args: -enablebonus

T8.in: 5 10 10 n l 2clo dro dro 2clo 3ri dro dro 2clo 6ri dro dro i clo 10ri dro dro l count 8ri dro RESTArt l 2clo dro dro 2clo 3ri dro dro 2clo 6ri dro dro s clo 10ri dro

T8 shows our enable bonus with more than 2 players, a smaller board, with some random commands.

T9 shows how the program runs when players loss and win

T9.args: nothing

T9.in: 100 dr I dr I 4ri dr I clo 8ri dr I clo 9ri dr i clo 10ri dr 100 dr y 100 dr 10ri dr lef ri ri dr 100dr Y 100 dr 10ri dr lef ri ri dr 100dr n

T9 shows how after one player loses, the other plays until the game ends, and the first person achieved the highest score wins. It also shows the hi score.

T10 shows how score are calculated when level is changed, and when single blocks in level 4 are be cleared

T10.args: -enablebonus

T10.in: 1 10 20 o dro o 2ri dro o 4ri dro o 6ri dro o 28levelu 8ri dro restart 1 10 13 o dro o 2ri dro o 4ri dro o 6ri dro 28levelu o 8ri dro dro dro clo 3ri dro dro dro dro 10ri dro count 7ri dro 23ri dro count 9ri dro clo 4ri dro count 5ri dro

T10 shows how the score is calculated in different cases. Before restart, Os are all generated before we change the level, all O have level 0, there are two lines that are cleared so score=$(4+2)^2+5*(0+1)^2=41$. After restart, the last O is generated after we change the level, there are two lines that are cleared so score=$(4+2)^2+4*(0+1)^2+1*(4+1)^2=65$. Then we make 5 consecutive calls without clearing any line, a block in the center left is dropped (since width is an even number), and another 5 consecutive calls without clearing any line, another block in the center is dropped. Then we clear the second lowest line, the single block is cleared so we add that score as well, so the score is $65+(4+1)^2+1*(4+1)^2=115$.