Nathan Chen

UID: 604902470

Project 3 Report

I didn’t make any functions virtual or pure virtual because there is no inherited code; I hardcoded every single different type of piece.

Piece Class:

* getType() returns the assigned PieceType of the constructed Piece object, stored in private member m\_type. I chose for this to be in the Piece class because it’s an accessor function for one of the Piece’s private variables.
* getRepresentation() returns the instructions to create a character representation of the Piece, assigned according to the PieceType of the Piece when the object is constructed. ‘#’ represents a ‘#’ character on the screen, ‘n’ represents a new line, and ‘ ‘ represents the blank space ‘ ‘ on the screen. I chose for this to be in the Piece class because it’s an accessor function for one of the Piece’s private variables.
* getOrientation() returns the orientation number of the Piece object, stored in private member orientation. All Pieces start in orientation 0, and rotate from 0 to 1 to 2 to 3 back to 0. I chose for this to be in the Piece class because it’s an accessor function for one of the Piece’s private variables.
* incrementOrientation() increments the orientation number of the Piece from 0 to 1 to 2 to 3 back to 0, and this function is used whenever the rotate() function successfully rotates a piece. I chose for this to be in the Piece class because it’s a mutator function for one of the Piece’s private variables.
* rotate() rearranges the ‘#’ characters making up the piece according to the orientation that it is supposed to be rotated to and depending on what PieceType it is. If rotating the piece would cause a collision with the walls of the well or with other objects in the well, the piece is not rotated. I chose for this to be in the Piece class because it’s a function that modifies the character representation of a piece on the screen.
* foamExplode() fills out a 5x5 area in the grid centered on where the foam bomb falls, as long as the paths from the center to the areas to be filled are unobstructed by other pieces or the well. I chose for this to be in the Piece class because it’s a function that modifies the character representation of a piece on the screen.
* vaporExplode() removes blocks from two rows above and two rows under where the vapor bomb drops, as well as from where the vapor bomb landed, as long as those areas are within the well. I chose for this to be in the Piece class because it’s a function that modifies the character representation of a piece on the screen.

Game Class:

* play() clears the screen, displays the well, displays/updates the status (score, rows left, level, next piece), runs levels until the game is lost, and displays prompts according to where the game is going. It tells the player to press the Enter key to begin playing, press the Enter key to advance to the next level if the current one is passed, and press the Enter key if the level is failed and they should exit the game.
* playOneLevel() drops random pieces down the well at certain intervals of time (if possible) and calls the rotate() function if the up arrow key is pressed. If the level is failed or the game is quit by pressing ‘q’ or ‘Q,’ the function returns false. Otherwise, it returns true once the level is completed (by filling a certain number of rows depending on the level). The piece is shifted to the left or right or bottom if the left or right or bottom arrow keys are pressed and the piece does not overlap with the well or other pieces by doing so, respectively. Once the piece is no longer movable, the score, rows filled, rows left to fill, and next piece are updated. In addition, filled rows are vaporized on the screen and pieces above are shifted down accordingly.
* displayStatus() displays the score, rows left to fill, current level, and next piece, displaying the values of the totScore and m\_level private members in the Game class and using the m\_level and rowsFilled private members to calculate and display the rows left to fill.
* displayPrompt() displays instructions for the player (i.e. press enter to advance a level, start the game, or quit the program) in the same spot on the screen, overwriting previous prompts.

Well Class:

* display() displays the left, bottom, and right walls of the well with ‘@’ characters, without interfering with the grid representation in the Game class.

I completed all functionality and don’t know of any bugs in my classes. It was ambiguous what to do if more rows are filled than necessary to pass a level (e.g. only one row left to fill but a piece completes 3 rows), so I let the display show negative levels to let the player know how many extra rows have been filled.