Zane Koch

404984674

CS32 Spring 2018

Project 3 Report

**Table of contents:**

-Page 1-11: description of public member function

-Page 11: description of missing functionality

-page 11: description of assumptions

**High level description of each public member function:**

Class Game:

Game(): This is the only game constructor. It takes two int arguments of width and height and using them constructs all of Game’s data members: 2 well objects, a screen object, level, score and rows destroyed. This is in this class out of necessity.

Void Play(): This function is called by a game piece to start a game. It serves as a pseudo ‘main’ function to control all operations from the game. It calls Game::playOneLevel to start each new level when the next one finishes, or if the game is over it destroys the pieces, prints the proper end game message and returns, ending the game. It is also the location from which the display functions that show the well are called. This is in this class out of necessity because of the way the main was written for us. It is not virtual because nothing inherits from game.

Bool playOneLevel(): This function is called in Game::play() to start each new level by creating a new piece and dropping it at a certain speed, depending on the value of m\_level. It keeps dropping new pieces once the piece stops moving until either the number of rows necessary to move onto the next level is fulfilled or the well is full. It returns true and false respectively in these scenarios to tell game::play() to either start a new level or end the game. This is in game because game is the control center for the movement of pieces and display of well, so it would not make sense for it to be anywhere else. It is not virtual because nothing inherits from game.

Void displayPrompt(string): This function display the string it is input at the default location for prompts. In game because it needs no information from any other class. It is not virtual because nothing inherits from game.

Void displayStatus(): This function display the prompts of score, nextpiece, rows left and level at their appropriate locations. In game because it needs information from both piece and well to propogate up before it can display the values. It is not virtual because nothing inherits from game.

Int checkDeleteRow(well, int yval): This function, given a certain well, always the well that the pieces are in, goes row by row checking if they are full. If a row is full then it removes the row. It recursively calls itself until it is at the bottom line of the well where it returns an integer that is the number of rows it deleted. This is inside of playOneLevel every time a new piece comes to rest. Its integer output determines the score. In game because it needs information from both piece and well to propogate up before it can display the values. It is not virtual because nothing inherits from game.

Void displayScoreAndRowsAndLevel(): this function display the score, row, and level at the appropriate place on the screen. It is called each time any of these values changes. It is in game because it only rarely needs to be called but needs information from other classes to propogate up before it has the correct numbers.

Well:

Well(): this is the only constructor for well. It initializes its member 2D array to all 0’s so that well::display does not display anything for them. This is in well out of necessity and is not virtual because nothing inherits from it.

Void Display(screen, int x, int y): this function display the border of the well that is made up of ‘@’s. It uses the coordinates given to it to write the correct number of rows and columns. It is only called once at the beginning of the game because the ‘@’s never needs to be redisplayed. This is in well because it logically makes sense for class well to display the well and is not virtual because nothing inherits from it.

Void displayStationary(screen): this function displays any value in the 2D member array, that represents the interior of the well, as a ‘$’. This member array is updated by piece whenever a piece stops moving. Display stationary is called everytime a new piece stops on the bottom, or when a row is full and gets deleted. This is in well because it display information about the well and is not virtual because nothing inherits from it.

Int getStationaryPieces(int x, int y): This is a getter function that allows other classes to see where in the well is occupied by a stationary piece. This is useful so moving pieces can know to stop when they are on top of a stationary piece. This is in well because it display information about the well and is not virtual because nothing inherits from it.

Void setStationaryPieces(int x, int y): this is a setter function that allows other classes to change what locations in well are occupied by stationary pieces. This is useful in cases where a row gets deleted because it is full. . This is in well because it changes information about the well and is not virtual because nothing inherits from it.

Void ClearStationaryPieces(): this is a function that sets all the values in the Well 2D member array, that represents the interior of the well, to 0 so they are not displayed as anything. This is used between levels to empty the well for the next level. This is in well because it changes information about the well and is not virtual because nothing inherits from it.

Void displayEmpty(Screen screen): this is very similar to clearStationaryPieces, but instead of actually emptying the well it just displays the well as if it was empty. This is only used at the very end of the game when the well must just look like it is empty but does not actually need to be emptied. This is in well because it displays information about the well and is not virtual because nothing inherits from it.

Piece:

Piece(int pounds): this is the piece constructor. It sets the value of pieces member variables based on the number of #’s, AKA locations, the piece takes up. This constructor is only only every called inside the constructor of one of the derived classes of Piece because piece is an abstract base class. It is in piece out of necessity and is not virtual because it does not need to be overridden.

wouldOverlap(Well well): this function determines if, when a piece moves down one row, it would run into another piece or exit the bounds of the well. It accesses the occupied values in the well and check if they are full before allowing the piece to move down. It returns true it the piece and move and false if not. It is not virtual because every piece has the same rules for overlapping other pieces or exiting the well. It is in the piece class because every piece must be able to check if it is allowed to move and so being in this location expedites that process.

wouldOverlapRight(Well well): this function does the same thing as wouldOverlap but checks if the value to the right is occupied instead of value below. This is called when the user inputs a keystroke that is intepretted as moving the piece to the right. It is not virtual because every piece has the same rules for overlapping other pieces or exiting the well. It is in the piece class because every piece must be able to check if it is allowed to move and so being in this location expedites that process.

wouldOverlapLeft(Well well): this function is identical to wouldOverlapRight except it checks to the left.

Bool timeToMove(int level, Well well): this function is called inside of a Piece’s move function. Bases on the level it waits a certain amount of time before allowing the pieces to move down. During this time it takes input from the keyboard to make the piece move in the specified direction. It is not virtual because every piece moves in the same timing given a certain level. It is in the piece class because everypiece must be able to call it easily so it can check if it is time for itself to move.

Void display(): this function displays the moving piece based on the value’s in piece’s 2D 11X18 array. This array is filled with 0’s initially but when a piece is added, the locations the piece occupies are filled with 1’s. Display then interprets these 1’s and displays a ‘#’ at their locations to show a moving piece. It is not virtual because every piece is displayed in the same way so there is no need for a piece to override it. It is in piece because every piece must be able to display itself while it is moving, having it anywhere else would be illogical.

Virtual bool move(int level, Well& well): this function move a piece. It first calls timeToMove and waits for it return. Once it has returned it called wouldOverlap. If would overlap is false it means it is safe for the piece to move so the 2D array is updated by moving all of the 1’s down one location. If wouldOverlap returns true it means the piece should come to rest so it is not moved and everyplace where there was a 1, well’s 2D array is given a 2 so ‘$’s will be displayed. This function is virtual because everypiece moves the same except the crazy piece which moves in the opposite direction. It being virtual allows crazy piece to override it but also does not make all the other pieces implement their own move. It is in piece because all pieces move and so they should all inherit this function.

Void moveRight(well): this does the same things as move, except instead of moving down, it moves to the right. It is called inside of is time to move in the userinputs a keystroke that indicated moving right. It also calls wouldoverlapright to make sure it can move in that direction. It is not virtual because all pieces move right in the same way so it should be inherited to all pieces. The crazy piece calls this function when keyboard input indicates it to traditionally move left. It is in piece so all derived classes of piece are able to move right in the same way.

Void moveLeft( well): is identical to moveright but moves left instead.

Bool SpacebarMove(Well well): This function is called if the user presses the spacebar. It continuously makes timeToMove return true, making the piece move down one row, until wouldOverlap returns true. It basically expedites the moving down process by taking out the delay between each movement and thus instantaneously moves the piece to the bottom of the well or wherever it should come to rest. This is in Piece, and not virtual, because all pieces move down in the same way, so all pieces should also move down in the same way when the spacebar is hit.

Virtual void rotate(well well) = 0: This function is called when the uses inputs an up arrow to switch the piece from its current orientation to its next orientation in a cyclical manner. This function is pure virtual because every piece rotates in a very similar, but at least in my implementation, non-standardized way. Each piece has its own implementation of this that looks through the 2D array inside of Piece and whenever it finds a 1 it moves it to the next location that it should be so the piece is displayed in its next orientation. This is in piece because everypiece must implement a rotate function, even if it does nothing like in the case of the foam bomb, because the spec lists them all as having 4 different orientations.

Int getPounds() const: this is a getter function that returns the number of pounds/spaces a piece has/takes up. This is used in the move function to differentiate the one and two location occupying pieces from the four location occupying pieces. This is not virtual, and is in piece, so that every piece’s number of #’s can be got.

Int getM\_X() and int getM\_Y(): these two function return the member variables of m\_x and m\_y respectively. This is useful so when a new piece needs to be displayed, it can locate itself in relation to the upperleft corner of its bounding box. This is not virtual, and is in piece, so that every piece’s number of m\_x and m\_y can be got.

Int getSpaceOccupied( int x, int y): this is a getter function that returns the value and a certain x,y location in the 2D array inside of piece. It is needed so that derived classes of piece can search through this array to find themselves and move themselves through the move(), moveRight() and moveLeft() functions. This is not virtual, and is in piece, so that every piece’s inherits it and its 2D array can be observed.

Int getOrientation(): this is a getter function that returns what orientation the piece is in, based off the private member variables m\_orientation, so each piece knows what orientation to go to next. This is not virtual, and is in piece, so that every piece inherits it and its orientation can be observed.

Screen getScreen(): this is a getter that allows derived pieces to get and subsequently display to the same screen. This is not virtual, and is in piece, so that every piece inherits it and its screen can be written to.

Void setSpaceOccupied(int x, int y, int newnum): this function allows all pieces to change the values that they occupy in the 2D array that represents the moving piece inside of Piece. It is needed because this is a private member variable of Piece and so its derived classes cannot change it without this setter. This is not virtual, and is in piece, so that every piece inherits it and its 2D can be changed in accordance with its movement.

Void setOrientaiton(): this function increments a pieces orientation after the orientation has been changed when the user inputs an up arrow equivalent. This is not virtual, and is in piece, so that every piece inherits it and its orientation number can be changed in accordance with its movement.

Bool setPieceOccupiedIfNotOccupied(int x, int y, int newnum, well well): this is a function that is used when a piece is trying to rotate. This function checks if, when the piece would rotate, it would go out of the bounds of the well or overlap another piece. If this is true it will not let to piece rotate. This is used everytime a piece tries to rotate for every type of piece so it is not virtual, and is in piece, so that every piece inherits it and it is possible to check if a piece should be allowed to move to its next orientation.

Bool wellFullGameOver(Well well): this function check if it is impossible for the next piece dropped to not overlap a current piece in the well. If this is the case it returns a bool telling playOneLevel that the game is over and the well should be clear etc. It is called everytime a new piece is about to be dropped to see if it will fit onto the board. For this reason it is not virtual, and is in piece, so that every piece inherits it and it is possible to check if a piece should be allowed to be dropped into the well or not.

Virtual ~piece(): this is an empty virtual destructor of piece. It is virtual so when a derived Piece is destructed the Piece part of it is also destructed. It is in piece so every piece is destructed.

T\_piece:

T\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In t\_piece out of necessity, not virtual because nothing inherits from t\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in t\_piece as a virtual function to indicate that it was a virtual function within Piece.

L\_piece:

L\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In L\_piece out of necessity, not virtual because nothing inherits from j\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in L\_piece as a virtual function to indicate that it was a virtual function within Piece.

J\_piece:

j\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In j\_piece out of necessity, not virtual because nothing inherits from j\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in j\_piece as a virtual function to indicate that it was a virtual function within Piece.

o\_piece:

o\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In o\_piece out of necessity, not virtual because nothing inherits from o\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in o\_piece a virtual function to indicate that it was a virtual function within Piece.

s\_piece:

s\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In s\_piece out of necessity, not virtual because nothing inherits from s\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in s\_piece as a virtual function to indicate that it was a virtual function within Piece.

z\_piece:

z\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In z\_piece out of necessity, not virtual because nothing inherits from z\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in z\_piece as a virtual function to indicate that it was a virtual function within Piece.

i\_piece:

i\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In i\_piece out of necessity, not virtual because nothing inherits from i\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in i\_piece as a virtual function to indicate that it was a virtual function within Piece.

crazy\_piece:

crazy\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In crazy\_piece out of necessity, not virtual because nothing inherits from crazy\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in crazy\_piece as a virtual function to indicate that it was a virtual function within Piece.

Bool timeToMove(int level, well well): this is an override of Piece’s time to move that switches what happens when the user inputs equivalent of left or right arrow key. This makes it move right when the left arrow is pressed and visa versa for left arrow. This is in crazy\_piece as a virtual function to indicate that it was a virtual function within Piece. This is in crazy\_piece because all crazy pieces move in the exceptional way.

Bool move( int level, well& well): this is an override of the piece move function that instead of calling the normal piece function’s time to move, it calls the timeToMove override in crazy\_piece so it moves in the opposite direction. This is in crazy\_piece as a virtual function to indicate that it was a virtual function within Piece. This is in crazy\_piece because all crazy pieces move in the exceptional way.

vapor\_piece:

vapor\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In vapor\_piece out of necessity, not virtual because nothing inherits from vapor\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in vapor\_piece as a virtual function to indicate that it was a virtual function within Piece.

crazy\_piece:

foam\_piece(): constructor that sets each initial location of a #, based on the spec, to a 1 in the 2D array in Piece so the information can be accessed later and the #’s can be moved. Also calls the constructor of Piece with an argument of 4 to set all of pieces data members per the Piece(int) constructor. In foam\_piece out of necessity, not virtual because nothing inherits from foam\_piece.

Rotate(well): takes a well as an argument, and depending on the value of orientation, when it located a 1 in the m\_spacesoccupied 2D array in Piece, moves the 1 to a new location corresponding to the next orientation. This is in foam\_piece as a virtual function to indicate that it was a virtual function within Piece.

**Missing implementations**

1. The implementation of Foam\_piece is missing. All it does is drop in a single # with no special functionality. I could not figure out the algorithm for recursively placing \*’s in a 5X5 box. I am pretty sure I could have implemented it using a stack, like in the maze solver, but time has run out.
2. The next piece does not display. I did not have time to figure out how to implement this but my pseudo-code for it would have been running the random piece creator function twice and storing one in a temp that get displayed under next piece while the other is dropped. Then the temp would get dropped and the random would be run again to get the next, next to be dropped.
3. Lastly the press q to quit functionality does not work. This is because I implemented playOneLevel recursively, so returning from it is kind of difficult because the return must propogate multiple levels up. I could not figure out how to have user input in the timeToMove function, where user input is collected, return all the way back to the playOnelevel function.

**Design Decisions, assumptions**

1. I was not sure how to “right justify” the score and rows left etc so it is simply identical to the example executable in that respect, however this may change as the score increases or something like that.
2. It was ambiguous how fast pieces had to fall when the space bar was pressed. Mine does not immediately re-display at the bottom of the well, instead it displays it as each new place it can be moved, but still goes to the bottom very quickly.