**Project 3 Report**

by Jamie Liu #204981426

1. High Level Descriptions:

Game

**Game(int width, int height);**

The constructor for this class was provided in the skeleton.

**void play();**

This function was provided in the skeleton.

**bool playOneLevel();**

This function allows the user to play a single level.

**void displayPrompt(std::string s);**

This function was provided in the skeleton.

**void displayStatus();**

This function displays the information about the game, including the score, the number of rows remaining, and the current level.

**int spaces(int digits);**

This function returns the number of spaces required to right-justify the numbers for the score, number of rows remaining, and current level.

**void displayNextPiece(PieceType type);**

This function displays the next piece to be played.

Well

**Well();**

The Well's constructor constructs a Well by initializing its member variables.

**~Well();**

The Well's destructor deletes the Well's dynamically allocated Piece.

**void display(Screen& screen, int x, int y) const;**

This function displays the well by calling displayBorder(screen, x, y) and displayWell(screen, x, y). It is defined in Well so that a Game can ask its Well to draw itself on the screen.

**void displayBorder(Screen& screen, int x, int y) const;**

This function draws the border of the well. It is defined in Well so that the Well's display method may call it.

**void displayWell(Screen& screen, int x, int y) const;**

This function draws the contents of the well. It is defined in Well so that the Well's display method may call it.

**void clearWell();**

This function empties the well by setting each of its values to the space character. It is defined in Well so that a Game can ask its Well to empty itself.

**bool isEmpty(int x, int y) const;**

This function returns true if the x and y coordinates provided are in bounds and if the Well's value at those coordinates is ' ' or '#'. It is defined in Well so that a Piece may ask its Well if certain coordinates are legal to move to.

**void createPiece(PieceType type);**

This function dynamically allocates a Piece of PieceType type by using the Well's m\_piece pointer. It is defined in Well so that a Game can ask its Well to create a Piece.

**int getX() const;**

This function returns the x coordinate of m\_piece's 4x4 bounding box. It is defined in Well in order to simplify the code in Well's other member functions.

**int getY() const;**

This function returns the y coordinate of m\_piece's 4x4 bounding box. It is defined in Well in order to simplify the code in Well's other member functions.

**PieceType getT() const;**

This function returns the PieceType of m\_piece. It is defined in Well in order to simplify the code in Well's other member functions.

**int getO() const;**

This function returns the orientation of m\_piece. It is defined in Well in order to simplify the code in Well's other member functions.

**void destroyPiece();**

This function deallocates m\_piece, sets m\_piece to nullptr, and sets the Well's m\_inPlay variable to false. It is defined in Well in order to allow a Game to ask its Well to deallocate its dynamically allocated memory and update its state.

**void drawPiece();**

This function adds the Well's Piece into its m\_well array. It is defined in Well so that a Game can ask its Well can update its contents.

**bool inPlay() const;**

This function returns true if there is currently a Piece in play. It is defined in Well so that a Game can ask its Well if there is a Piece in play or not.

**bool atBottom() const;**

This function returns true if the Piece in play cannot move any lower in the Well at its current position. It is defined in Well so that a Game can ask its Well if a Piece has reached the bottom or not.

**void movePiece(char ch);**

This function moves the Well's Piece in the indicated direction if the desired movement is valid. It is defined in Well so that a Game can ask its Well to move its Piece.

**void rotatePiece();**

This function rotates the Well's Piece once if the rotation is a valid movement. It is defined in Well so that a Game can ask its Well to rotate its Piece.

**void performAction(PieceType type);**

This function accepts a PieceType and executes the appropriate action that the Piece should take when it lands, depending on the type. (i.e. for a foam piece it calls foam(), for a vapor piece it calls vapor(), and for a regular piece it calls fill()). It is defined in Well so that a Game may ask its Well to perform actions when a Piece lands.

**void fillPaths(char arr[5][5], int x, int y);**

This function accepts a 5x5 character array where 'e' denotes empty and 'w' denotes a wall, and starting from the xy coordinate given, fills each coordinate that is reachable with an 'f' character. It is defined in Well so that the Well's foam() method may call it.

**void foam();**

This function creates foam as indicated by the project specification by filling each reachable coordinate in the 5x5 grid centered at the foam piece with foam. It is defined in Well so that the Well's performAction method may call it.

**void vaporize();**

This function vaporizes the vapor piece and the 2x2 grids above and below the vapor piece as indicated by the project specification. It is defined in Well so that the Well's performAction method may call it.

**void fill();**

This function converts a piece into '$' characters to indicate that the piece has come to rest. It is defined in Well so that the Well's performAction method may call it.

**bool isFull(int y) const;**

This function returns true if the row at y-coordinate y is a full row. It is defined in Well so that the Well's removeLine method may call it.

**void removeLine(int y);**

This function removes the line at the y-coordinate y and moves all the rows above it down by 1. It is defined in Well so that a Game may ask its Well to remove a full line.

**bool isGameOver() const;**

This function returns true if the game is over (i.e. if the Well's Piece is in an illegal configuration). It is defined in Well so that a Game may ask its Well if the game should be over or not.

Piece

**Piece(PieceType type);**

The constructor for Piece constructs a Piece by setting its xy coordinates, orientation, and type.

**int getOrientation() const;**

This function returns the current orientation of the Piece. It is defined in Piece so that a Well can get its Piece's orientation.

**bool setOrientation(int orientation);**

This function sets the Piece's orientation. It is defined in Piece so that a Piece can change its orientation.

**int getX() const;**

This function returns the x coordinate of the upper left corner for the Piece's 4x4 bounding box. It is defined in Piece so that a Well can get its Piece's x coordinate.

**int getY() const;**

This function returns the y coordinate of the upper left corner for the Piece's 4x4 bounding box. It is defined in Piece so that a Well can get its Piece's y coordinate.

**void setX(int x);**

This function sets the x coordinate of the upper left corner for the Piece's 4x4 bounding box. It is defined in Piece so that a Well can set its Piece's x coordinate.

**void setY(int y);**

This function sets the y coordinate of the upper left corner for the Piece's 4x4 bounding box. It is defined in Piece so that a Well can set its Piece's y coordinate.

**PieceType getType() const;**

This function returns the Piece's PieceType. It is defined in Piece so that a Well can get its Piece's PieceType.

**virtual void move(char ch);**

This function moves the Piece in the indicated direction by changing its x and y coordinates. It is defined in Piece so that a Well can call it on any piece. I made this function virtual because a Crazy piece moves differently than all of the other pieces.

**void rotate();**

This function rotates the Piece by changing its orientation once. It is defined in Piece so that a Well can rotate its Piece.

**bool isABlock(PieceType type, int x, int y, int orientation) const;**

This function returns true if the xy coordinate in the Piece's 4x4 bounding box with the given PieceType and orientation has a piece block. It is defined in Piece so that a Well or Game can determine where its Piece's blocks are.

**virtual bool okToMove(const Well\* well, char ch) const;**

This function returns true if the Piece can legally move in the indicated direction in the given Well. It is defined in Piece so that a Well can ask its Piece if it can move in a certain direction. I made this function virtual because the Crazy piece moves differently from all of the other pieces, so it must have a different okToMove implementation.

**virtual bool okToRotate(const Well\* well) const;**

This function returns true if the Piece can legally rotate in the given Well. It is defined in Piece so that a Well can ask its Piece if it can rotate. I made this function virtual because the pieces that do not have differences in their orientations will have implementations that simply return true.

PieceI

**PieceI();**

The constructor for PieceI initializes a PieceI making a Piece with type PIECE\_I.

PieceL

**PieceL();**

The constructor for PieceL initializes a PieceL making a Piece with type PIECE\_L.

PieceJ

**PieceJ();**

The constructor for PieceJ initializes a PieceJ making a Piece with type PIECE\_J.

PieceT

**PieceT();**

The constructor for PieceI initializes a PieceT making a Piece with type PIECE\_T.

PieceO

**PieceO();**

The constructor for PieceO initializes a PieceO making a Piece with type PIECE\_O.

**virtual bool okToRotate(Well\* well) const;**

This function returns true. I defined it in PieceO in order to specialize Piece's version of okToRotate, which returns based on the Piece's type and orientation. I made it virtual in order to enable polymorphism and allow Piece pointers to access it.

PieceS

**PieceS();**

The constructor for PieceS initializes a PieceS making a Piece with type PIECE\_S.

PieceZ

**PieceZ();**

The constructor for PieceZ initializes a PieceZ making a Piece with type PIECE\_Z.

PieceVapor

**PieceVapor();**

The constructor for PieceVapor initializes a PieceVapor making a Piece with type PIECE\_VAPOR.

**virtual bool okToRotate(Well\* well) const;**

This function returns true. I defined it in PieceVapor in order to specialize Piece's version of okToRotate, which returns based on the Piece's type and orientation. I made it virtual in order to enable polymorphism and allow Piece pointers to access it.

PieceFoam

**PieceFoam();**

The constructor for PieceFoam initializes a PieceFoam making a Piece with type PIECE\_FOAM.

**virtual bool okToRotate(Well\* well) const;**

This function returns true. I defined it in PieceO in order to specialize Piece's version of okToRotate, which returns based on the Piece's type and orientation. I made it virtual in order to enable polymorphism and allow Piece pointers to access it.

PieceCrazy

**PieceCrazy();**

The constructor for PieceCrazy initializes a PieceCrazy making a Piece with type PIECE\_CRAZY.

**virtual void move(char ch);**

This function moves the PieceCrazy by in the indicated direction by changing its x and y coordinates. It is defined in PieceCrazy in order to specialize Piece's version of move, because a PieceCrazy's left and right instructions are reversed. I made it virtual in order to enable polymorphism and allow Piece pointers to access it.

2. Incomplete Functionality:

To my understanding, my game exhibits all required functionality.

3. Design Decisions and Assumptions:

1. I decided to implement the well as a 2D array.
2. I decided to use a 4D array to store all of the different pieces' orientations.
3. I decided that each Well would contain a pointer to a Piece.
4. I decided that a Game should also have member variables to keep track of the score and number of rows left.
5. I decided to not make Piece an abstract base class, because I wanted to avoid having each subclass of Piece keep track of its own PieceType.
6. I thought that the spec (combined with the playable executable) was very thorough with regards to what to do in all situations, so I did not need to make any assumptions about the intended behavior.