Brandon Wolfson

Project 3 Report

1.

Game

```
Game(int width, int height);
```
This creates a game object which has all the necessary data and methods to play the game, with a well of passed width and height. Constructors are class-local.

```
void play();
```
This plays a game of Chetyris. I defined it in Game because you play a Game, you do not play a Well or a Piece. I did not make this virtual as Game has no derived classes.

Well

```
Well(int width, int height);
```
This constructs a well of passed width and height. Constructors are class-local.

```
~Well();
```
This destructs a well, deallocating any dynamic variables the well has allocated. I did not make this virtual because there are no derived classes of Well.

```
void display(Screen& screen, int x, int y) const;
```
This displays the well on the screen so the user can see it. I defined this here because a Well object contains a data structure representing the game board as well as pieces on the board, so it can easily print itself out. I did not make this virtual because there are no derived classes of Well.

```
bool readyForNextPiece() const;
```
This returns true if there is not a piece currently falling in the well, or false otherwise. A Well object knows if a piece is currently falling in it, so it makes sense to put this method in the Well class. I did not make this virtual because there are no derived classes of Well.

```
bool addPiece(PieceType current);
```
This adds a new piece to the well. I defined this in the Well class because wells control what the game board looks like, so adding a piece would be a method that you'd ask the Well object to do. I did not make this virtual because there are no derived classes of Well.

```
bool movePiece(char direction);
```
This moves the current piece either left or right on the well, if possible. I defined this method in the Well class because moving the current piece changes what the game board looks like, so the Well object would have to update its representation of the game board. I did not make this virtual because there are no derived classes of Well.

```
bool rotatePiece();
```
This rotates the current piece, if possible. I defined this method in the Well class because rotating the current piece changes what the game board looks like, so the Well object would have to

update its representation of the game board. I did not make this virtual because there are no derived classes of Well.

```
int updateAfterOneTimeUnit();
```

This moves the current piece down one unit, or makes the current piece come to rest if it would overlap with the well or something in the well if it moved down. I defined this method in the Well class because moving the current piece downwards or changing the current piece to '$' to represent that it has come to rest changes what the game board looks like, so the Well object would have to update its representation of the game board. I did not make this virtual because there are no derived classes of Well.

```
void emptyWell();
```

This erases all pieces currently in the well, leaving an empty well. I defined this method in the Well class because it changes what the game board looks like. I did not make this virtual because there are no derived classes of Well.

Piece

```
Piece(PieceType piece);
```

This creates a piece of the passed PieceType. I defined this in the Piece class because all pieces have a set of orientations, and this constructor creates the orientations corresponding to the passed PieceType. Constructors are class-local and never virtual.

```
virtual ~Piece();
```

This destructs the Piece data members. I defined this in the Piece class because all pieces must be destructed at some point. I made this virtual because VaporBomb and FoamBomb are derived classes from Piece, and thus, to avoid undefined behavior and to ensure that VaporBomb's and FoamBomb's destructor is called, it is necessary to make the base class destructor virtual.

```
void prevOrientation();
```

This changes a piece to have its previous orientation. I defined this in the Piece class because all pieces can be set to their previous orientation. I did not make this virtual because no piece has a special way of going to its previous orientation.

```
void nextOrientation();
```

This changes a piece to have its next orientation. I defined this in the Piece class because all pieces can be set to their next orientation. I did not make this virtual because no piece has a special way of going to its next orientation.

```
void moveLeft();
```

This changes a piece's location to be one unit left. I defined this in the Piece class because all pieces have a location and must be able to move left. I did not make this virtual because no piece has a special way of moving left.

```
void moveRight();
```

This changes a piece's location to be one unit right. I defined this in the Piece class because all pieces have a location and must be able to move right. I did not make this virtual because no piece has a special way of moving right.

```
void moveDown();
```

This changes a piece's location to be one unit down. I defined this in the Piece class because all pieces have a location and must be able to move down. I did not make this virtual because no piece has a special way of moving down.

```
int getOrientation() const;
```

This returns the piece's current orientation. I defined this in the Piece class because all pieces must be able to return their current orientation. I did not make this virtual because no piece has a special way of returning their orientation.

```
int getX() const;
```

This returns the piece's current x-coordinate. I defined this in the Piece class because all pieces must be able to return their current x-coordinate. I did not make this virtual because no piece has a special way of returning their current x-coordinate.

```
int getY() const;
```

This returns the piece's current y-coordinate. I defined this in the Piece class because all pieces must be able to return their current y-coordinate. I did not make this virtual because no piece has a special way of returning their current y-coordinate.

```
char getBox(int row, int col) const;
```

This returns character in the passed row and column of a piece's bounding box. I defined this in the Piece class because all pieces must be able to return the characters of their bounding box. I did not make this virtual because no piece has a special way of returning the characters of their bounding box.

```
virtual bool performSpecialAction(vector<vector<char>>& board) const;
```

This performs a piece's special action (like the special actions of VaporBomb and FoamBomb) or returns false if the piece does not have a special action. I defined this in the Piece class because my Well object used a Piece pointer, so this method had to be present in the base class to call it in my derived classes (VaporBomb and FoamBomb). I made this virtual because normal pieces do not have any special actions, while VaporBombs and FoamBombs do have different special actions. I did not make this pure-virtual because my normal pieces were instances of the Piece class, so I decided not to make my Piece class an Abstract Base Class.

VaporBomb

```
VaporBomb()
```

This constructs a VaporBomb piece. I defined this in the VaporBomb class because constructors are class-local and never virtual.

```
virtual bool performSpecialAction(vector<vector<char>>& board) const;
```

This performs VaporBomb's special action, which is vaporizing blocks 2 tiles above and below the vapor bomb. I defined this in the VaporBomb class because only vapor bombs have this

special action. I made this virtual because it is good style to use the virtual keyword in your derived classes if you've labeled the function virtual in your base class.

FoamBomb

```
FoamBomb()
```

This constructs a FoamBomb piece. I defined this in the FoamBomb class because constructors are class-local and never virtual.

```
virtual bool performSpecialAction(vector<vector<char>>& board) const;
// Fills all reachable, open spaces within a 5x5 bounding box
//     surrounding the foam bomb into *'s.
```

This performs FoamBomb's special action, which is filling reachable empty spaces within a 5x5 box surrounding the foam bomb in the well. I defined this in the FoamBoms's class because only foam bombs have this special action. I made this virtual because it is good style to use the virtual keyword in your derived classes if you've labeled the function virtual in your base class.

2.

All functionalities have been completed and there are no known bugs in the program.

3.

I decided to make my normal pieces (i.e. all pieces except VaporBomb and FoamBomb) objects of my Piece class and only created derived classes for the VaporBomb and FoamBomb pieces. This was because the only difference between normal pieces was they had different bounding box orientations, and only VaporBombs and FoamBombs required actual different methods. I decided it made more sense just to have all normal pieces be Piece objects and to construct their bounding box orientations in Piece's constructor according to whatever PieceType was being constructed.