Stratego Refactoring Project

Milestone 2

CSSE 375

Software Construction and Evolution

3/30/2016

Logan Erexson, Jason Lane

Jacob Knispel, Steve Trotta

Iteration Achievements

The main achievement for this week was the major redesign of the system. While not what the client may want to hear, we are proud of the progress we have made. Most of the key features rely on GamePiece or some such representation being created and the redesign that comes inherently with such a data model change.

For example, KillFeed needs to know which piece killed which; this which would previously have required using (at least) a relation of integers to Strings representing piece names and teams. Similarly, implementing piece animation would have been almost impossible without an abstract GamePiece class, because the graphical interface would have no way of tracking which pieces are in the middle of movement since they were not objects.

Previously, the addition of more pieces meant that new conditions or case statements must be added across all methods that require piece-type knowledge (which was previously solely integer based). This would require an extreme amount of work, more so in the long run than our current redesign job (even if it takes us the rest of the quarter). And lastly, plans such as modifiers and runtime behavior changes of pieces and their rules would be impossible with the currently designed system. So, not only did we have to refactor, which we did, but we had to redesign - this became one of our features for the milestone.

Creating Behavior objects that can be applied to Piece objects is also a large improvement to the project, as piece code no longer needs to be duplicated throughout the project, reducing future and current code duplication. This also means that piece functionality can change at runtime, which was not previously possible.

As for other achievements, we got quite a bit of refactoring done, as well as starting on revamping the game's AI player. We also implemented a rough version of the kill feed feature and began work on the piece movement feature. It looks like we are mostly in a good place to be able to present our new features by Friday to the client despite the cost of our major redesign.

Code Smell Refactoring

List of smell fixes below:

Steve - backPanel's paint method	(switch statement)
Steve - Movepiece	(long method and comments)
Jason - BoardPosition class	(data clumps)
Jason - Lazy BuffPanel class	(lazy class)
Jacob - Gameboard class	(primitive obsession)
Jacob - Save/load extraction	(large class)
Logan - Field piece placing	(feature envy)
Logan - StrategoGame class	(large class)

Switch Statement in backPanel_Paint(object sender, PaintEventArgs e) (Steve Trotta) :

```
if (piece != 42)
{
    if (piece > 0)
    {
        // Piece is on the blue team, so we change the brush color to blue
        //b = new SolidBrush(Color.FromArgb(25, 25, 15 * Math.Abs(piece)));
        b = new SolidBrush(Color.FromArgb(25, 25, 175));
        pen.Color = Color.FromArgb(200, 200, 255);
    }
    else
    {
        // Piece is on the red team, so we change the brush to reflect that
        //b = new SolidBrush(Color.FromArgb(15 * Math.Abs(piece), 25, 25));
        b = new SolidBrush(Color.FromArgb(175, 25, 25));
        pen.Color = Color.FromArgb(255, 200, 200);
    }

    int cornerX = x * col_inc + paddingX;
    int cornerY = y * row_inc + paddingY;
    Rectangle r = new Rectangle(x * scaleX + (scaleX - (int)(scaleY * .55)) / 2, y * scaleY + 5, (int)(scaleY *
    switch(piece)
    {
        case 9:
            // Piece is a blue scout (displaying as image)
            if (turn == 1 || this.lastFought.Equals(new Point (x, y)))
            {
                Image imag = Properties.Resources.BlueScout;
                e.Graphics.DrawImage(imag, r);
                if (this.pieceIsSelected && this.pieceSelectedCoords.X == x && this.pieceSelectedCoords.Y == y)
                    pen.Color = Color.FromArgb(10, 255, 10);
                g.DrawRectangle(pen, r);
            }
            else
            {
                g.DrawRectangle(pen, r);
                g.FillRectangle(b, r);
            }
            break;
        case -9:
            // Piece is a red scout (displaying as image)
            if (turn ==-1 || this.lastFought.Equals(new Point(x, y)))
            {
                Image imag = Properties.Resources.RedScout;
                e.Graphics.DrawImage(imag, r);
                if (this.pieceIsSelected && this.pieceSelectedCoords.X == x && this.pieceSelectedCoords.Y == y)
                    pen.Color = Color.FromArgb(10, 255, 10);
                g.DrawRectangle(pen, r);
            }
            else
            {
                g.FillRectangle(b, r);
                g.DrawRectangle(pen, r);
            }
```

Smells: Switch Statement and Multiple, Nested Conditionals

This paint method initialized pen colors, images, and other drawable objects depending on (manually) the negative or positive values of the integer representation of the game piece and the absolute value of the game piece. These caused a multi-page case statement and duplication of code as well as multiple, nested conditionals and an excessively long method.

Current Solution (at the time of writing):

```
public GamePiece(int teamCode)
{
    this.pieceRank = 42;
    this.pieceName = "null";
    this.teamCode = teamCode;
    this.lifeStatus = true;
    this.pieceImage = null;
    this.pieceColor = (teamCode == StrategoGame.BLUE_TEAM_CODE) ? BLUE_TEAM_COLOR : RED_TEAM_COLOR;
    this.attackBehavior = new DefaultComparativeFate();
    this.defendBehavior = new DefaultComparativeFate();

    this.limitToMovement = 1;

    this.movable = true;
    this.xVal = -1;
    this.yVal = -1;

    this.essential = false;
}
```

New GamePiece representation of game pieces

```
public void addPieceToFactory(String identifier, int numberRef, Type pieceType)
{
    this.stringDict.Add(identifier, pieceType);
    this.intDict.Add(numberRef, pieceType);
}

public GamePiece getPiece(String identifier, int teamCode)
{
    if (identifier.Equals(GamePiece.NULL_PIECE_NAME))
    {
        return null;
    }
    Type type = this.stringDict[identifier];

    var ctors = type.GetConstructors();

    return (GamePiece)ctors[0].Invoke(new object[] { teamCode });
}
```

New GamePieceFactory code to generate a desired GamePiece

```
for (int x = 0; x < boardState.getWidth(); x++)
{
    for (int y = 0; y < boardState.getHeight(); y++)
    {
        int scaleX = this.panelWidth / boardState.getWidth();
        int scaleY = this.panelHeight / boardState.getHeight();
        GamePiece piece = boardState.getPiece(x, y);
        if(piece!=null && piece.getTeamCode()!=0)
        {
            Brush b = new SolidBrush(piece.getPieceColor());
            pen.Color = Color.FromArgb(200, 200, 255);
            // pen.Color = Color.FromArgb(255, 200, 200);

            int cornerX = x * col_inc + paddingX;
            int cornerY = y * row_inc + paddingY;
            Rectangle r = new Rectangle(x * scaleX + (scaleX - (int)(scaleY * .55)) / 2, y * scaleY + 5, (int)(scaleY * .55), scaleY - 10);

            if (this.game.turn == piece.getTeamCode() || boardState.getLastFought()!=null && boardState.getLastFought().Equals(new Point(x, y)))
            {
                Image imag = piece.getPieceImage();
                e.Graphics.DrawImage(imag, r);
                if (piece == selectedGamePiece)
                {
                    pen.Color = Color.FromArgb(10, 255, 10);
                }
                g.DrawRectangle(pen, r);
            }
            else
            {
                g.DrawRectangle(pen, r);
                g.FillRectangle(b, r);
            }
            if (pieceMoves[x, y] == 1)
            {
                r = new Rectangle(x * scaleX + 1, y * scaleY + 1, scaleX - 2, scaleY - 2);
                //b.Color = Color.FromArgb(90, 90, 255);
                g.FillRectangle(new SolidBrush(Color.FromArgb(100, 130, 130, 130)), r);
            }
        }
```

The new and improved backPanel_Paint is much shorter, because the Image is retrieved from the piece, the color is chosen from the piece and the only conditionals needed are the to check for the piece last fought and that the correct team's pieces are allowed to be seen on the board / are revealed.

The piece itself is of the class GamePiece. This class determines what color and image it is depending on the team code and contains identifiers that allow for better rule management. The creation of attack and defend methods that are inherited by subclasses of GamePiece that either keep or override them - this is helpful in that special rules are extracted from the rest of the logic, generally making the code less dense and conditionally intricate. The GamePieceFactory allows a unified way to extract appropriate types of game pieces using reflection and dictionaries to avoid confusion that was inherent and rampant in using integer arrays.

Smells: Long Method and Comments (Steve Trotta) :

```
/// <summary>
/// Moves the selected piece(if there is one) to the tile tile which corresponds to the x,y coords (if valid)
/// </summary>
/// <param name="x">x coordinate of the mouse click of where to move (pixels)</param>
/// <param name="y">y coordinate of the mouse click of where to move (pixels)</param>
/// <returns>true if a piece was moved, false otherwise</returns>
public bool MovePiece(int x, int y)
{
    int scaleX = this.panelWidth / this.boardState.GetLength(0);
    int scaleY = this.panelHeight / this.boardState.GetLength(1);
    if (!this.pieceIsSelected)
        return false;
    this.pieceIsSelected = false;
    if (Piece.attack(this.boardState[this.pieceSelectedCoords.X, this.pieceSelectedCoords.Y],
        this.boardState[x / scaleX, y / scaleY]) == null)
        return false;
    if (Math.Abs(this.boardState[this.pieceSelectedCoords.X, this.pieceSelectedCoords.Y]) != 9)
    {
        if (Math.Abs((x / scaleX) - this.pieceSelectedCoords.X) > 1 || Math.Abs((y / scaleY) - this.pieceSelectedCoords.Y) > 1)
            return false;
    }
    else
    {
        //Check for the scout's special cases
        if(Math.Abs((x / scaleX) - this.pieceSelectedCoords.X) > 1)
        {
            if (((x / scaleX) - this.pieceSelectedCoords.X) > 1)
            {
                for(int i = 1; i < (x / scaleX) - this.pieceSelectedCoords.X; i++)
                {
                    if(this.boardState[this.pieceSelectedCoords.X+i, this.pieceSelectedCoords.Y] != 0)
                        return false;
                }
            }
            else if (((x / scaleX) - this.pieceSelectedCoords.X) < -1)
            {
                for(int i = -1; i > (x / scaleX) - this.pieceSelectedCoords.X; i--)
                {
                    if(this.boardState[this.pieceSelectedCoords.X+i, this.pieceSelectedCoords.Y] != 0)
                        return false;
                }
            }
        }
        else if (Math.Abs((y / scaleY) - this.pieceSelectedCoords.Y) > 1)
        {
            if (((y / scaleY) - this.pieceSelectedCoords.Y) > 1)
            {
                for (int i = 1; i < (y / scaleY) - this.pieceSelectedCoords.Y; i++)
                {
                    if (this.boardState[this.pieceSelectedCoords.X, this.pieceSelectedCoords.Y + i] != 0)
                        return false;
                }
```

```
                else if (((y / scaleY) - this.pieceSelectedCoords.Y) < -1)
                {
                    for (int i = -1; i > (y / scaleY) - this.pieceSelectedCoords.Y; i--)
                    {
                        if (this.boardState[this.pieceSelectedCoords.X, this.pieceSelectedCoords.Y + i] != 0)
                            return false;
                    }
                }
            }
        }
        if (Math.Abs((x / scaleX) - this.pieceSelectedCoords.X) >= 1 && Math.Abs((y / scaleY) - this.pieceSelectedCoords.Y) >= 1)
            return false;
        if (Math.Abs((x / scaleX) - this.pieceSelectedCoords.X) == 0 && Math.Abs((y / scaleY) - this.pieceSelectedCoords.Y) == 0)
            return false;
        int defender = this.boardState[x / scaleX, y / scaleY];
        this.boardState[x / scaleX, y / scaleY] = Piece.attack(this.boardState[this.pieceSelectedCoords.X, this.pieceSelectedCoords.Y],
        if ((defender == 0) || this.boardState[x / scaleX, y / scaleY]==0)
            this.lastFought = new Point(-1, -1);
        else
            this.lastFought = new Point(x / scaleX, y / scaleY);
        this.boardState[this.pieceSelectedCoords.X, this.pieceSelectedCoords.Y] = 0;
        if (defender == 12)
        {
            gameOver(-1);
        }
        else if (defender == -12)
        {
            gameOver(1);
        }
        else { this.nextTurn(); }
        return true;
    }
```

Current Solution:

```
        /// <summary>
        /// Moves the selected piece(if there is one) to the tile tile which corresponds to the x,y coords (if valid)
        /// </summary>
        /// <param name="x">x coordinate of the mouse click of where to move (pixels)</param>
        /// <param name="y">y coordinate of the mouse click of where to move (pixels)</param>
        /// <returns>true if a piece was moved, false otherwise</returns>
        public bool MovePiece(int x, int y)
        {
            GamePiece defender = this.boardState.getPiece(x, y);
            if (this.selectedGamePiece == null)
            {
                return false;
            }
            else if (this.selectedGamePiece.getXVal() == x && this.selectedGamePiece.getYVal() == y)
            {
                // Initialize "Selection Phase"
                this.selectedGamePiece = null;
                return false;
            }
            Move move = new Stratego.Move(this.selectedGamePiece.getXVal(), this.selectedGamePiece.getYVal(), x, y);
            bool res = this.boardState.move(move);

            if (!boardState.isGameOver())
            {
                this.nextTurn();
            }
            this.selectedGamePiece = null;
            return res;
        }
```

```
public void attack(GamePiece otherPiece)
{
    if(this.attackBehavior.decideFate(this, otherPiece))
    {
        this.killPiece();
    }
}

public void defend(GamePiece otherPiece)
{
    if(this.defendBehavior.decideFate(this, otherPiece))
    {
        this.killPiece();
    }
}
```

attack() and defend() are for attacking and defending other objects - they are inherited or overridden by subclasses of GamePiece.

Because of the aforementioned attack and defend and GamePiece relationships, we have an easy method to battle pieces and the changing of pieces from place to place is simplified by have a board of GamePieces that each contain their own X and Y for reference. We can also change our logic for what beats what and who can move where using BattleBehaviors and the allowable number of spaces a piece can move. This gives runtime flexibility to our system and will allow for us to better and more easily create a KillFeed, add GamePieces, and change the general behavior of the game without hardcoded values or overly complex conditionals.

```
[TestFixture()]
public class PieceTest
{
    public static readonly int BOTH_DEAD = 99;
    public static readonly int FAILURE = 100;

    //Marshal = 1, General = 2, Colonel = 3, Major = 4, Captain = 5, Lieutenant = 6, Sergeant = 7, Miner = 8, Cout = 9, Spy = 10, Bomb
    public static IEnumerable<TestCaseData> SpyVSSpyData
    {
        get
        {
            GamePiece newSpy0 = new SpyPiece(0);
            GamePiece newSpy1 = new SpyPiece(1);
            yield return new TestCaseData(newSpy0, newSpy1);
        }
    }

    [TestCaseSource("SpyVSSpyData")]
    public void TestSpyVSpyBattle(GamePiece a, GamePiece b)
    {
        Assert.AreEqual(BOTH_DEAD, returnExpectedOnAttack(a, b));
        GamePiece attack2Spy = new SpyPiece(0);
        GamePiece defend2Spy = new SpyPiece(1);
        Assert.AreEqual(BOTH_DEAD, returnExpectedOnAttack(attack2Spy, defend2Spy));
    }

    [TestCase()]
    public void TestSpyVDefaultBattle()
    {
        GamePiece a1Spy = new SpyPiece(0);
        GamePiece d1Scout = new ScoutPiece(1);
```

```
    [TestCase()]
    public void TestSpyVMinerBattle()
    {
        GamePiece a1Spy = new SpyPiece(0);
        GamePiece d1Miner = new MinerPiece(1);
        Assert.AreEqual(MinerPiece.MINER_RANK, returnExpectedOnAttack(a1Spy, d1Miner));
        GamePiece a2Miner= new MinerPiece(0);
        GamePiece d2Spy = new SpyPiece(1);
        Assert.AreEqual(MinerPiece.MINER_RANK, returnExpectedOnAttack(a2Miner, d2Spy));
    }

    [TestCase()]
    public void TestSpyVFlagBattle()
    {
        GamePiece a1 = new SpyPiece(0);
        GamePiece d1 = new FlagPiece(1);
        Assert.AreEqual(SpyPiece.SPY_RANK, returnExpectedOnAttack(a1, d1));
        GamePiece a2 = new FlagPiece(0);
        GamePiece d2 = new SpyPiece(1);
        Assert.AreEqual(BOTH_DEAD, returnExpectedOnAttack(a2, d2));
    }

    [TestCase()]
    public void TestSpyVMarshallBattle()
    {
        GamePiece a1 = new MarshallPiece(0);
        GamePiece d1 = new SpyPiece(1);
        Assert.AreEqual(MarshallPiece.MARSHALL_RANK, returnExpectedOnAttack(a1, d1));
        GamePiece a2 = new SpyPiece(0);
        GamePiece d2 = new MarshallPiece(1);
        Assert.AreEqual(SpyPiece.SPY_RANK, returnExpectedOnAttack(a2, d2));
    }
```

I rewrote the old test cases, checking better for boundary cases while reducing the overwhelming nature of the many different cases that were necessary when int arrays needed to be tested. There is a one-to-one correlation to the previous tests and these, so whatever passed as the logic for the previous system, in regards to piece battling, still exists.

BoardPosition (Jason Lane) - Before

```
public bool MovePiece(int x, int y)
{
    int scaleX = this.panelWidth / this.boardState.GetLength(0);
    int scaleY = this.panelHeight / this.boardState.GetLength(1);
    if (!this.pieceIsSelected)
        return false;
    this.pieceIsSelected = false;
    if (Piece.attack(this.boardState[this.pieceSelectedCoords.X, this.pieceSelectedCoords.Y],
        this.boardState[x / scaleX, y / scaleY]) == null)
        return false;
    if (Math.Abs(this.boardState[this.pieceSelectedCoords.X, this.pieceSelectedCoords.Y]) != 9)
    {
        if (Math.Abs((x / scaleX) - this.pieceSelectedCoords.X) > 1 || Math.Abs((y / scaleY) - this.pieceSelectedCoords.Y) > 1)
            return false;
    }
    else
    {
        //Check for the scout's special cases
        if(Math.Abs((x / scaleX) - this.pieceSelectedCoords.X) > 1)
        {
            if (((x / scaleX) - this.pieceSelectedCoords.X) > 1)
            {
                for(int i = 1; i < (x / scaleX) - this.pieceSelectedCoords.X; i++)
                {
                    if(this.boardState[this.pieceSelectedCoords.X+i, this.pieceSelectedCoords.Y] != 0)
                        return false;
                }
            }
            else if (((x / scaleX) - this.pieceSelectedCoords.X) < -1)
            {
                for(int i = -1; i > (x / scaleX) - this.pieceSelectedCoords.X; i--)
                {
                    if(this.boardState[this.pieceSelectedCoords.X+i, this.pieceSelectedCoords.Y] != 0)
                        return false;
                }
            }
        }
        else if (Math.Abs((y / scaleY) - this.pieceSelectedCoords.Y) > 1)
        {
            if (((y / scaleY) - this.pieceSelectedCoords.Y) > 1)
            {
                for (int i = 1; i < (y / scaleY) - this.pieceSelectedCoords.Y; i++)
```

Smell: Data Clumps

Anything to do with the boardstate or piece layout had x and y grouped together, and generally kept the same pairs together consistently.

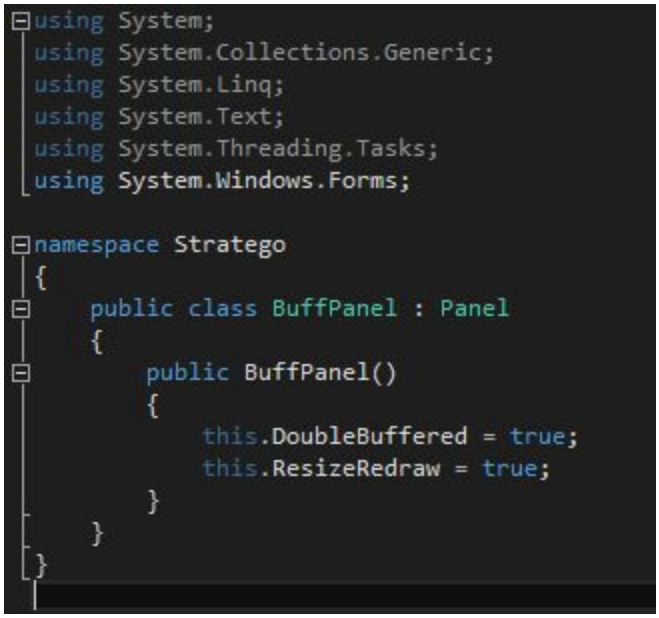
BoardPosition - After

```
namespace Stratego
{
    [Serializable]
    public class BoardPosition
    {
        public static readonly BoardPosition NULL_BOARD_POSITION = new BoardPosition(-1, -1);

        private int x;
        private int y;
        public BoardPosition(int x, int y)
        {
            this.x = x;
            this.y = y;
        }

        public int getX()
        {
            return this.x;
        }

        public int getY()
        {
            return this.y;
        }

        public bool isNull()
        {
            return this.Equals(BoardPosition.NULL_BOARD_POSITION);
        }

        public override bool Equals(Object other)
        {
            if (other == null)
                return false;

            if (other.GetType() != this.GetType())
                return false;

            BoardPosition otherBP = (BoardPosition)other;
            return this.x == otherBP.getX() && this.y == otherBP.getY();
        }
    }
}
```

Solution: Extract a BoardPosition class that would hold these parameters together. While in some cases the new class is less convenient to use than the original code, in general it is at least as usable as the original. Additionally, it enables the use of BoardPositions as parameters, removing the need for multiple functions to take the x and y coordinates of a piece separately.

Test Cases:
Because BoardPosition's only responsibility is to simplify the existing code, it is covered by the test cases for the code blocks that now use it (specifically those for Gameboard and StrategoGame, which are shown in later parts of the document).

BuffPanel (Jason Lane) - Before

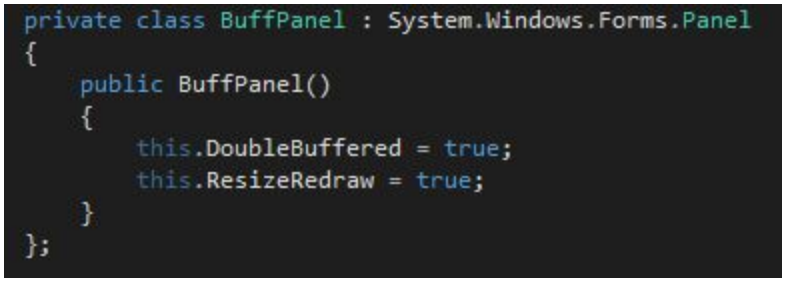


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Stratego
{
    public class BuffPanel : Panel
    {
        public BuffPanel()
        {
            this.DoubleBuffered = true;
            this.ResizeRedraw = true;
        }
    }
}
```

Smell: Lazy Class

BuffPanel existed as a separate file for the sole purpose of extending panel and setting the DoubleBuffered field. Running everything in a panel with these attributes prevents the background from flickering uncontrollably, so it's very necessary to have around. However, having something this simple in its own class clutters the project.

BuffPanel - After



```
private class BuffPanel : System.Windows.Forms.Panel
{
    public BuffPanel()
    {
        this.DoubleBuffered = true;
        this.ResizeRedraw = true;
    }
};
```

Solution: Make BuffPanel a private class internal to Main.Designer.cs (the only class that used it) so that it doesn't clutter up the namespace.

Test Cases: BuffPanel's purpose was to prevent the background from flickering. Due to the graphical nature of the solution, it was altogether unnecessary/impossible to write test cases that check whether or not the background of the window flickers.

Gameboard (Jacob Knispel) - Before

```
public int[,] boardState { get; set; }

...

int num_cols = this.boardState.GetLength(0);
int num_rows = this.boardState.GetLength(1);


...


boardState[9-k, 3-j] = turn*Convert.ToInt32(numbers[k]);

...

public void fillRow(int value, int row)
{
	for (int x = 0; x < this.boardState.GetLength(0); x++) this.boardState[x, row] = value;
}
```

Smell: Primitive obsession

The reason the above implementation is poor is because it complicated the class boardState was in significantly. Whenever the board needed to be reset (which is fairly often), the function fillRow() needs to be called, which isn't really relevant to the class it is in and clutters up the namespace. Our obsession with primitives resulted in being forced to pass around an unwieldy 2D array instead of a compact, succinct object.

Gameboard - After

```
[Serializable]
public class Gameboard
{
    private int width;
    private int height;
    private BoardPosition lastFought;
    private int winner = 0;
    private GamePiece[,] board;

    public Gameboard(int width, int height)
    {
        resetBoard(width, height);
    }

    public void resetBoard()
    {
        // Board should be filled with null automatically (empty spaces)
        this.board = new GamePiece[this.width, this.height];
    }

    public void resetBoard(int width, int height)
    {
        this.width = width;
        this.height = height;
        resetBoard();
    }

    private void gameOver(int v)
    {
        this.winner = v;
    }

    private void battlePieces(GamePiece attacker, GamePiece defender)
    {
        attacker.attack(defender);
        defender.defend(attacker);
        if (!defender.isAlive())
```

Solution: Our solution was to create the Gameboard class, which contains a 2D array of Piece objects (a result of a redesign we completed this milestone). This class also holds any methods relevant to the board, so that this class can be passed about and acted upon with simple getWidth() and getHeight() methods included instead of complicated and confusing calls to retrieve simple information. It is also much easier to redefine the board with a different width and height with this implementation, since our Gameboard class has a simple resetBoard() function, whereas previously, resetting the board required reinstantiating an array, which was awkward.

Test cases:

Many test cases use the game's board, and so there are slews of tests that verify that the Gameboard class functions in the same way, although some tests needed tweaking:

```
[TestCase(2, 3)]
[TestCase(6, 2)]
[TestCase(2, 205)]
public void TestGameBoard(int gbW, int gbH)
{
    Gameboard g = new Gameboard(gbW, gbH);

    Assert.AreEqual(gbW, g.getWidth());
    Assert.AreEqual(gbH, g.getHeight());

    Assert.AreEqual(null, g.getPiece(0, 0));
    g.setPiece(0, 0, new SpyPiece(1));
    Assert.AreEqual(SpyPiece.SPY_RANK, g.getPiece(0, 0).getPieceRank());

    g.fillRow(null, 0);

    Assert.AreEqual(null, g.getPiece(0, 0));
}
```

Saving (Jacob Knispel) - Before

```
/// <summary>
/// A trigger that activates when the SaveButton in the main menu is pressed
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SaveButton_Click(object sender, EventArgs e)
{
    SaveFileDialog dialog = new SaveFileDialog();
    dialog.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
    string path = System.IO.Path.GetDirectoryName(Application.ExecutablePath);
    if (path.EndsWith("\\bin\\Debug") || path.EndsWith("\\bin\\Release"))
    {
        for (int i = 0; i < path.Length - 3; i++)
        {
            if ((path[i] == '\\') && (path[i + 1] == 'b') && (path[i + 2] == 'i') && (path[i + 3] == 'n'))
            {
                path = path.Substring(0, i);
                break;
            }
        }
    }
    dialog.InitialDirectory = System.IO.Path.Combine(path, @"Resources\SaveGames");
    dialog.RestoreDirectory = true;

    if(dialog.ShowDialog() == DialogResult.OK)
    {
        StreamWriter writer = new StreamWriter(dialog.FileName);
        saveGame(writer);
        writer.Close();
    }
}
```

```
/// <summary>
/// Saves a gamestate into the string or file in the given writer
/// </summary>
/// <param name="writer"></param>
/// <returns> True if successful </returns>
public bool saveGame(TextWriter writer)
{
    if ((this.turn == 0)||(this.preGameActive)||(Math.Abs(this.turn)==2)) return false;

    string buffer = "";
    if (isSinglePlayer)
        buffer = " 1 "+this.ai.difficulty;
    else
        buffer = " 0";
    writer.WriteLine(this.turn + buffer);
    for (int i = 0; i < boardState.GetLength(1); i++ )
    {
        buffer = "";
        for (int j = 0; j < boardState.GetLength(0)-1; j++)
        {
            buffer += boardState[j, i] + " ";
        }
        buffer += boardState[boardState.GetLength(0) - 1, i];
        writer.WriteLine(buffer);
    }

    return true;
}
```

Smell: Large class
(Very low cohesion)

Originally, save functionality was implemented through a lot of hardcoding and was intimately tied to the GUI since it was located in Main. It was difficult to test these functions because the GUI got in the way of our test code.

Saving - After

```
private static DialogResult displayFileDialog(FileDialog dialog, string extension)
{
    dialog.Filter = extension + " files (*." + extension + ")|*." + extension + "|All files (*.*)|*.*";
    dialog.InitialDirectory = System.IO.Path.Combine(System.IO.Path.GetDirectoryName(Application.ExecutablePath));
    dialog.RestoreDirectory = true;

    return dialog.ShowDialog();
}

public static void storeData(string fileName, Object data)
{
    IFormatter formatter = new BinaryFormatter();
    Stream stream = new FileStream(fileName, FileMode.Create, FileAccess.Write, FileShare.None);
    formatter.Serialize(stream, data);
    stream.Close();
}
```

```
public static bool saveGame(SaveData saveData)
{
    return saveSomething(saveData, SAVE_FILE_EXTENSION);
}

private static bool saveSomething(Object data, string fileExtension)
{
    FileDialog dialog = new SaveFileDialog();

    if (displayFileDialog(dialog, fileExtension) == DialogResult.OK)
    {
        storeData(dialog.FileName, data);
        return true;
    }

    return false;
}
```

Solution: In order to fix this smell, we extracted methods for save and load out of the giant class into a new class called SaveLoadOperations that handles any saving/loading operations that are distinct from the GUI. Now our main window invokes SaveLoadOperations in order to accomplish its tasks. We also changed saving so that it uses BinarySerialization instead of saving values directly to a text file, which took up a lot of space.

Test Cases:

These test cases verify that the SaveData that is saved is equal to the SaveData that is loaded back in.

```
[TestFixture()]
class SaveLoadTests
{
    [TestCase(2, 3, 2, 1, true)]
    [TestCase(6, 2, 4, -1, false)]
    [TestCase(2, 205, 5, 0, true)]
    [TestCase(100, 30, 2, 1, true)]
    [TestCase(6, 2, 3, 4, false)]
    [TestCase(2, 25, 51, 0, true)]
    public void TestSaveLoad(int gbW, int gbH, int difficulty, int turn, bool isSinglePlayer)
    {
        SaveData saveData = new SaveData(new Gameboard(gbW, gbH), difficulty, turn, true);

        SaveLoadOperations.storeData("test." + SaveLoadOperations.SAVE_FILE_EXTENSION, saveData);
        saveData = SaveLoadOperations.loadSaveData("test." + SaveLoadOperations.SAVE_FILE_EXTENSION);

        Assert.AreEqual(gbW, saveData.boardState.getWidth());
        Assert.AreEqual(gbH, saveData.boardState.getHeight());
        Assert.AreEqual(difficulty, saveData.difficulty);
        Assert.AreEqual(turn, saveData.turn);
        Assert.AreEqual(isSinglePlayer, saveData.isSinglePlayer);
    }
}
```

piecePlacing (Logan Erexson) - Before

```
public class StrategoGame
{

    /// <summary>
    /// The default amount of pieces for each piece. (
    /// </summary>
    public static readonly Dictionary<String, int> def
    { { FlagPiece.FLAG_NAME, 1 }, { BombPiece.BOMB_NAM
        { ScoutPiece.SCOUT_NAME, 8 }, { MinerPiece.MIN
        {LieutenantPiece.LIEUTENANT_NAME, 4 }, {Captai
        { GeneralPiece.GENERAL_NAME, 1}, { MarshallPie
    };

    public Dictionary<int, Type> pieceTypes = new Dict

    /// <summary>
    /// Current level of the game. Equals -1 if not in
    /// </summary>
    public int level { get; set; }

    /// <summary>
    /// The piece currently being placed by the user
    /// </summary>
    public GamePiece piecePlacing { get; set;}
```

```
private void SidePanelButtonClick(object sender, EventArgs e)
{
    // this.piecePlacing = Convert.ToInt32(((Button)sender).Text); No longer used, as I use the Tag text of the buttons instead.
    if (!this.removeCheckBox.Checked)
    {
        foreach (var button in this.SidePanel.Controls.OfType<Button>())
            button.UseVisualStyleBackColor = true;
        this.game.piecePlacing = this.factory.getPiece(Convert.ToInt32(((Button)sender).Tag), this.game.turn);
        ((Button)sender).UseVisualStyleBackColor = false;
    }
}

if (this.game.preGameActive)
{
     bool? piecePlaced = this.game.placePiece(this.game.piecePlacing, boardX, boardY);
```

```
private void removeCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (this.removeCheckBox.Checked)
    {
        //this.activeSidePanelButton = this.piecePlacing;
        this.activeSidePanelButton = 0;
        this.game.piecePlacing = this.factory.getPiece(0, this.game.turn);
    }
    else
    {
        this.game.piecePlacing = this.factory.getPiece(this.activeSidePanelButton, this.game.turn);
    }
}
```

Smell: Feature Envy -
Multiple methods in StrategoWin request piecePlacing while no method in StrategoGame uses it.

piecePlacing - After

```
public partial class StrategoWin : Form, GUICallback
{

    /// <summary>
    /// An array that holds the different keys for the Konami code
    /// </summary>
    private Keys[] konami = new Keys[8] { Keys.Up, Keys.Up, Keys.D

    /// <summary>
    /// Stores how far through the Konami code the player has ente
    /// </summary>
    private int konamiIndex = 0;

    /// <summary>
    /// The piece currently being placed by the user
    /// </summary>
    public GamePiece piecePlacing { get; set; }
```

```
private void SidePanelButtonClick(object sender, EventArgs e)
{
    // this.piecePlacing = Convert.ToInt32(((Button)sender).Text); No longer used, as I use the Tag text of the buttons instead.
    if (!this.removeCheckBox.Checked)
    {
        foreach (var button in this.SidePanel.Controls.OfType<Button>())
            button.UseVisualStyleBackColor = true;
        this.piecePlacing = this.factory.getPiece(Convert.ToInt32(((Button)sender).Tag), this.game.turn);
        ((Button)sender).UseVisualStyleBackColor = false;
    }
}
```

```
if (this.game.preGameActive)
{
    bool? piecePlaced = this.game.placePiece(this.piecePlacing, boardX, boardY);

    // Only run if the placement succeeded
    if (piecePlaced.Value)
    {
```

```
private void removeCheckBox_CheckedChanged(object sender, EventArgs e)
{
    if (this.removeCheckBox.Checked)
    {
        //this.activeSidePanelButton = this.piecePlacing;
        this.activeSidePanelButton = 0;
        this.piecePlacing = this.factory.getPiece(0, this.game.turn);
    }
    else
    {
        this.piecePlacing = this.factory.getPiece(this.activeSidePanelButton, this.game.turn);
    }
}
```

Solution:

I moved piecePlacing from StrategoGame to StrategoWin.

Test Case:

```
public void TestStrategoWinHasPiecePlacing()
{
    StrategoWin win = new StrategoWin();
    GamePiece piece = new SpyPiece(1);
    win.piecePlacing = piece;
    Assert.AreEqual(piece, win.piecePlacing);
}
```

StrategoGame (Logan Erexson) - Before

```
14  namespace Stratego
15  {
16      public partial class StrategoWin : Form
17      {
18          /// <summary>
19          /// The default amount of pieces for each piece. (EX: 0 0s; 1 1; 1 2; 2 3s; 4 4s; etc..)
20          /// </summary>
```

```
public bool? placePiece(int piece, int x, int y)
{
    if (turn == 0 || Math.Abs(turn) == 2) return false;
    if (Math.Abs(piece) > 12 || x<0 || y<0 || x>this.panelWidth || y>this.panelHeight) throw new ArgumentException();
    if ((Math.Sign(piece) != Math.Sign(this.turn)) &&(piece!=0)) return false;
    Boolean retVal = true;
    int scaleX = this.panelWidth / this.boardState.GetLength(0);
    int scaleY= this.panelHeight / this.boardState.GetLength(1);
    int pieceAtPos = this.boardState[x / scaleX, y / scaleY];

    if (piece == 0 && pieceAtPos != 42)
    {
        // We are trying to remove
        if (Math.Sign(pieceAtPos) != Math.Sign(this.turn)) return false;
        if (pieceAtPos == 0) retVal = false;
        this.placements[Math.Abs(pieceAtPos)]++;
    }
    else if (pieceAtPos == 0 && this.placements[Math.Abs(piece)] > 0)
    {
        // We are trying to add
        this.placements[Math.Abs(piece)] -= 1;
    }
    else retVal = false;

    if (retVal) this.boardState[x / scaleX, y / scaleY] = piece;
    return retVal;
}
```

```
public void nextTurn()
{
    if(!testing)
        this.backPanel.Invalidate();

    // We just came here from the main menu
    if(this.turn == 0)
    {
        preGameActive = true;
        this.turn = 1;
    }
    // It's blue player's turn
    else if (this.turn == 1)
    {
        if (this.preGameActive)
        {
            this.turn = -1;
            this.placements = this.defaults;
        }
        else
        {
            this.turn = 2;
            if (!this.checkMoves())
                this.gameOver(1);
            else
            {
                if (!testing)
                {
                    if (!this.isSinglePlayer)
                        NextTurnButton.Text = "Player 2's Turn";
                    else
                        NextTurnButton.Text = "AI's Turn";
                    NextTurnButton.Visible = true;
                    this.NextTurnButton.Enabled = true;
                }
            }
        }
    }
```

```
public int[,] GetPieceMoves(int X, int Y, int[,] boardState)
{
    int[,] moveArray = new int[boardState.GetLength(1), boardState.GetLength(0)];
    if ((Math.Abs(boardState[X, Y]) == 0) || (Math.Abs(boardState[X, Y]) == 11 && !this.movableBombs) || (Math.Abs(boardState[X, Y]
        return moveArray;
    if (Math.Abs(boardState[X, Y]) == 9)
    {
        //for (int yD = Y + 1; yD < boardState.GetLength(1) && boardState[X, yD] == 0; yD++)
        //    moveArray[X, yD] = 1;
        //for (int yU = Y - 1; yU >= 0 && boardState[X, yU] == 0; yU--)
        //    moveArray[X, yU] = 1;
        //for (int xR = X + 1; xR < boardState.GetLength(0) && boardState[xR, Y] == 0; xR++)
        //    moveArray[xR, Y] = 1;
        //for (int xL = X - 1; xL >= 0 && boardState[xL, Y] == 0; xL--)
        //    moveArray[xL, Y] = 1;
        for (int yD = Y + 1; yD < boardState.GetLength(1) && ((Math.Sign(boardState[X, yD]) != Math.Sign(boardState[X, Y])) && boar
        {
            moveArray[X, yD] = 1;
            if ((Math.Sign(boardState[X, yD]) != Math.Sign(boardState[X, Y])) && (Math.Sign(boardState[X, yD]) != 0))
                break;
        }
        for (int yU = Y - 1; yU >= 0 && ((Math.Sign(boardState[X, yU]) != Math.Sign(boardState[X, Y])) && boardState[X, yU] != 42);
        {
            moveArray[X, yU] = 1;
            if ((Math.Sign(boardState[X, yU]) != Math.Sign(boardState[X, Y])) && (Math.Sign(boardState[X, yU]) != 0))
                break;
        }
        for (int xR = X + 1; xR < boardState.GetLength(0) && ((Math.Sign(boardState[xR, Y]) != Math.Sign(boardState[X, Y])) && boar
        {
            moveArray[xR, Y] = 1;
            if ((Math.Sign(boardState[xR, Y]) != Math.Sign(boardState[X, Y])) && (Math.Sign(boardState[xR, Y]) != 0))
                break;
        }
        for (int xL = X - 1; xL >= 0 && ((Math.Sign(boardState[xL, Y]) != Math.Sign(boardState[X, Y])) && boardState[xL, Y] != 42);
        {
            moveArray[xL, Y] = 1;
            if ((Math.Sign(boardState[xL, Y]) != Math.Sign(boardState[X, Y])) && (Math.Sign(boardState[xL, Y]) != 0))
                break;
        }
    }
    if (Y > 0)
        if ((Math.Sign(boardState[X, Y - 1]) != Math.Sign(boardState[X, Y])) && boardState[X, Y - 1] != 42)
            moveArray[X, Y - 1] = 1;
    if (Y < boardState.GetLength(1) - 1)
        if ((Math.Sign(boardState[X, Y + 1]) != Math.Sign(boardState[X, Y])) && boardState[X, Y + 1] != 42)
            moveArray[X, Y + 1] = 1;
    if (X < boardState.GetLength(0) - 1)
        if ((Math.Sign(boardState[X + 1, Y]) != Math.Sign(boardState[X, Y])) && boardState[X + 1, Y] != 42)
```

```
public bool? SelectPiece(int x, int y)
{
    if ((Math.Abs(turn) == 2)||(turn ==-1 &&isSinglePlayer)) return false;
    int scaleX = this.panelWidth / this.boardState.GetLength(0);
    int scaleY = this.panelHeight / this.boardState.GetLength(1);
    if ((this.pieceSelectedCoords == new Point(x / scaleX, y / scaleY))&&this.pieceIsSelected)
    {
        this.pieceIsSelected = false;
        return false;
    }
    if (((Math.Abs(this.boardState[x / scaleX, y / scaleY]) == 11 && !this.movableBombs) || (Math.Abs(this.boardState[x / scaleX, y
          Math.Sign(this.boardState[x / scaleX, y / scaleY]) != Math.Sign(this.turn))
    {
        return false;
    }
    this.pieceSelectedCoords = new Point(x/scaleX, y/scaleY);
    this.pieceIsSelected = true;
    return true;
}
```

Smell: Large Class -
The class StrategoWin was very large and contained a lot of logic to do with managing the game mixed with in with the GUI.

StrategoGame - After

```
public class StrategoGame
{

    /// <summary>
    /// The default amount of pieces for each piece. (EX: 0 0s; 1 1; 1 2; 2 3s; 4 4s; etc..)
    /// </summary>
    public static readonly Dictionary<String, int> defaults = new Dictionary<String, int>()
    { { FlagPiece.FLAG_NAME, 1 }, { BombPiece.BOMB_NAME, 6 }, { SpyPiece.SPY_NAME, 1 },
        { ScoutPiece.SCOUT_NAME, 8 }, { MinerPiece.MINER_NAME, 5 }, { SergeantPiece.SERGEANT_NAME, 4 },
        {LieutenantPiece.LIEUTENANT_NAME, 4 }, {CaptainPiece.CAPTAIN_NAME, 4 }, { MajorPiece.MAJOR_NAME, 3}, { ColonelPiece.COLONEL_NAME, 2},
        { GeneralPiece.GENERAL_NAME, 1}, { MarshallPiece.MARSHALL_NAME, 1 }
    };

    public Dictionary<int, Type> pieceTypes = new Dictionary<int, Type>();

    /// <summary>
    /// Current level of the game. Equals -1 if not in campaign mode
    /// </summary>
    public int level { get; set; }
```

```
public void nextTurn()
{
    this.callback.invalidateBackpanel();
    // We just came here from the main menu
    if (this.turn == NO_TEAM_CODE)
    {
        preGameActive = true;
        this.turn = BLUE_TEAM_CODE;
    }
    // It's blue player's turn
    else if (this.turn == BLUE_TEAM_CODE)
    {
        if (this.preGameActive)
        {
            this.turn = -1;
            this.resetPlacements();
        }
        else
        {
            this.turn = 2;
            if (!this.checkMoves())
                this.callback.gameOver(StrategoGame.BLUE_TEAM_CODE);
            else
            {
                if (!this.isSinglePlayer)
                    this.callback.adjustTurnButtonState("Player 2's Turn");
                else
                    this.callback.adjustTurnButtonState("AI's Turn");
            }
        }
    }
    // It's red player's turn
    else if (this.turn == RED_TEAM_CODE)
```

```
public bool? placePiece(GamePiece piece, int x, int y)
{
    if (turn == 0 || Math.Abs(turn) == 2) return false;
    if (piece != null && piece.getTeamCode() != turn) return false;
    Boolean retVal = true;

    GamePiece pieceAtPos = this.boardState.getPiece(x, y);

    if (piece == null)
    {
        // We are trying to remove

        if (pieceAtPos == null || pieceAtPos.getTeamCode() == NO_TEAM_CODE) return false;
        if (pieceAtPos.getTeamCode() != this.turn) return false;
        this.placements[pieceAtPos.getPieceName()]++;
    }
    else if (pieceAtPos == null && piece!=null && this.placements[piece.getPieceName()] > 0)
    {
        // We are trying to add
        this.placements[piece.getPieceName()] -= 1;
    }
    else retVal = false;

    if (retVal) this.boardState.setPiece(x, y, piece);
    return retVal;
}
```

```
public bool? SelectPiece(int x, int y)
{
    /*
        The if-block immediate below is NOT what we want, but haven't changed it yet to remind us
        to change the general strategy other places too. We'll need to come back and spot-check this stuff.
    */
    if ((Math.Abs(turn) == 2) || (turn == -1 && isSinglePlayer))
    {
        return false;
    }
    if (this.selectedGamePiece != null)
    {
        return true;
    }
    GamePiece potentialSel = this.boardState.getPiece(x, y);
    if (potentialSel == null)
    {
        this.selectedGamePiece = null;
        return false;
    }
    if ((!potentialSel.isMovable() || potentialSel.getTeamCode() != this.turn))
    {
        this.selectedGamePiece = null;
        return false;
    }
    this.selectedGamePiece = potentialSel;
    return true;
}
```

```
public int[,] GetPieceMoves(int x, int y, Gameboard boardState)
{
    int[,] moveArray = new int[boardState.getHeight(), boardState.getWidth()];

    GamePiece pieceInQuestion = boardState.getPiece(x, y);
    if (pieceInQuestion.isMovable() || pieceInQuestion.getLimitToMovement() == 0)
    {
        return moveArray;
    }
    int startingX = pieceInQuestion.getXVal();
    int startingY = pieceInQuestion.getYVal();
    int spacesPossible = pieceInQuestion.getLimitToMovement();
    GamePiece potenPiece = null;
    for (int k = startingX; k <= startingX + spacesPossible; k++)
    {
        if (k >= boardState.getWidth())
        {
            break;
        }
        potenPiece = boardState.getPiece(k, startingY);
        if (potenPiece == null || potenPiece.getTeamCode() == NO_TEAM_CODE || pieceInQuestion.getTeamCode() != potenPiece.getTeamCode())
        {
            break;
        }
        moveArray[k, startingY] = 1;
    }
    for (int i = startingX; i >= startingX - spacesPossible; i--)
    {
        if (i < 0)
        {
            break;
        }
        potenPiece = boardState.getPiece(i, startingY);
        if (potenPiece == null || potenPiece.getTeamCode() == NO_TEAM_CODE || pieceInQuestion.getTeamCode() != potenPiece.getTeamCode())
        {
            break;
        }
        moveArray[i, startingY] = 1;
    }
    for (int j = startingY; j <= startingY + spacesPossible; j++)
    {
        if (j >= boardState.getHeight())
```

Solution:

I moved code that had to do with game logic into a separate class called StrategoGame. Methods that referred to the GUI were altered so that they did not.

Test Cases:

```
[TestCase(1, 1, ExpectedResult = false)]
[TestCase(2, 2, ExpectedResult = false)]
[TestCase(3, 3, ExpectedResult = false)]
[TestCase(4, 4, ExpectedResult = false)]
// This tests that pieces will not be placed on an obstacle space
public bool? TestThatNothingCanBePlacedOnObstacle(int x, int y)
{

    StrategoGame game = new StrategoGame(new TestCallback());
    game.nextTurn();
    foreach(GamePiece piece in this.pieces)
    {
        if(game.placePiece(piece, x, y).Value)
        {
            return true;
        }
    }
    return false;
}
```