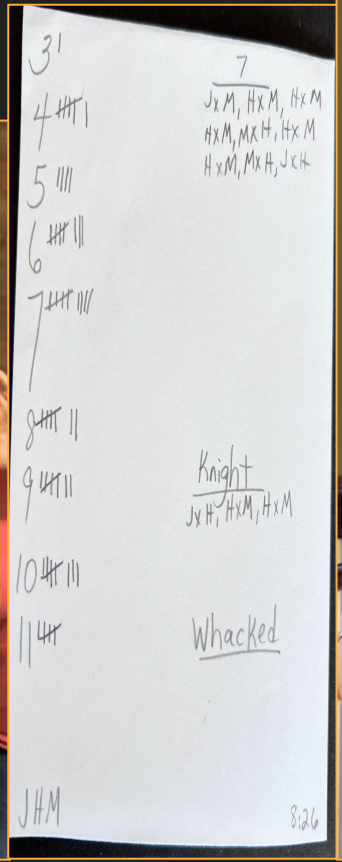
Catan Tracker

A Data Collection Application for Settlers of Catan

Reason for Development

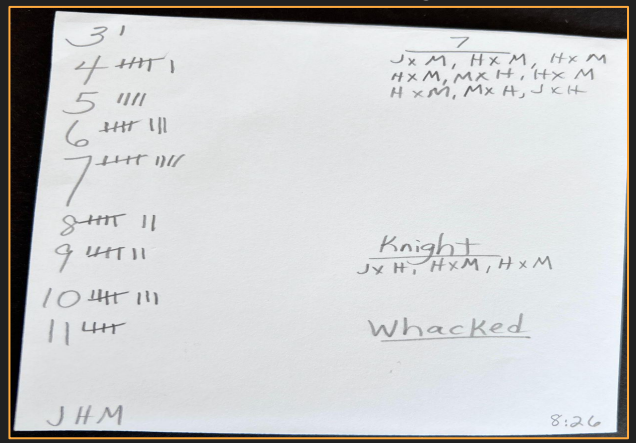
- My wife's family



Analog Tracked Data v. Digitally Tracked Data

ANALOG:

- Roll Results
- Times ‘Whacked’
- Knight Activity
- Robber Activity



DIGITAL: (All Analog Data PLUS)

- Turn Order
- Times Robbed
- Times Blocked
- Each Resource Earned
- Development Cards Bought
- Time per Turn
- Shortest Turn
- Longest Turn
- Average Turn Time
- (Any Other Metric Desired)

Basics of Settlers of Catan

- Roll Dice
- Acquire Resources
- Build
- Foil your Opponents
- Hexes/Tokens
- Largest Army
- Longest Road
- Settlements/Cities

Custom Classes

- GameBoard
  - Locations
  - Dice Rolls
  - Timers
- Player
  - Name
  - Event Counters
  - Settlements
  - Cities
  - Victory Points
- Location
  - Resource
  - Token
- Settlement
  - 3 Locations
  - Detection Methods
- City
  - 3 Locations
  - Detection Methods

GameBoard Class

- Available Board Locations
- Game Timer
- Count of Each Rolled Number
- Robber Blocking Location

```
internal class GameBoard {
    private int[] _brickPips;
    private int[] _goldPips;
    private int[] _orePips;
    private int[] _sheepPips;
    private int[] _wheatPips;
    private int[] _woodPips;
    private int _currentRound = 1;
    private int[] _rollCounter = new int[11];
    private int _blockedNumber = 0;
    private string _blockedResource = "";
    private Stopwatch _gameClock = new Stopwatch();
    private Stopwatch _turnClock = new Stopwatch();
    private Player _hasLargestArmy;
    private Player _hasLongestRoad;
    private Player _winner;

    private Location[] _hexArray;
    private Location _blockedLocation;
```

```
public GameBoard(Location[] hexArray) {
    _hexArray = hexArray;
}
```

Player Class

Object is representation of each player.

Stores Player-Specific Game Data:

- Place in Turn Order
- Each Resource Earned
- Settlements/Cities
- Victory Points
- Road/Army Size

```
internal class Player {
    private string _name;
    private bool _hasLargestArmy = false;
    private bool _hasLongestRoad = false;
    private bool _isWinner = false;
    private int _turnInGame;
    private int _settlements = 2;
    private int _cities = 0;
    private int _longestRoadLength = 1;
    private int _roadCount = 2;
    private int _stolenFromCount = 0;
    private int _knightCount = 0;
    private int _whackedCount = 0;
    private int _totalWood = 0;
    private int _totalBrick = 0;
    private int _totalGold = 0;
    private int _totalOre = 0;
    private int _totalSheep = 0;
    private int _totalWheat = 0;
    private int _timesBlocked = 0;
    private int _victoryPoints = 0;
    private int _devCardsBought = 0;
    private string _robberActivity = ""; //"Miles st
    private string _knightActivity = ""; //"Miles kn
    private TimeSpan[] _turnTimes;
    private bool[] _ownedNumbers = new bool[11]; //{
    private string[] _pipResources = new string[11];
    private Settlement[] _settlementObjects;
    private City[] _cityObjects;
```

```
internal class Location {
    private int _tokenNum;
    private string _resource;

    1 reference
    public Location(int tokenNum, string resource) {
        _tokenNum = tokenNum;
        _resource = resource;
    }
    25 references
    public int Token {
        get {return  _tokenNum; } set { _tokenNum = value; }
    }

    32 references
    public string Resource {
        get {return _resource;} set { _resource = value; }
    }

    4 references
    public string LocationToString() {
        string response = $"{_resource}-{_tokenNum}";
        return response;
    }
}//END CLASS
```

```
25 references
internal class Settlement {
    private Location _hex1 = null;
    private Location _hex2 = null;
    private Location _hex3 = null;

    2 references
    public Settlement(Location location1, Location location2 = null, Location location3 = null) {
        _hex1 = location1;
        _hex2 = location2;
        _hex3 = location3;
    }
```

```
11 references
internal class City {
    private Location _hex1 = null;
    private Location _hex2 = null;
    private Location _hex3 = null;

    2 references
    public City(Settlement settlement) {
        Location[] hexes = settlement.HexArray;
        _hex1 = hexes[0];
        _hex2 = hexes[1];
        _hex3 = hexes[2];
        settlement = null;
    }
```

Location is a simple class, but effectively stores the Token/Resource pairing.

Settlement and City are almost identical classes. They store the location data affiliated with their placement.

3 Distinct Phases

- PRE-GAME
  - Set up the Board
  - Initialize the Players
- MAIN GAME LOOP
  - Handle Dice Roll Outcomes
  - Handle Build Options
  - Handle and Aggregate All Data
- END-GAME
  - Collect Data
  - Display Data
  - Offer Export Option for Long-Term Storage

PRE-GAME : Handling the Initial Board Setup

- Create a GameBoard class
- Create a Location class (w/ Token and Resource properties)
- Have user input the token/resource pairings
- Save each token/resource pairing as a Location object


```
<Grid>
    <Grid Name="viewBoardSetup" Visibility="Collapsed">
        <Image HorizontalAlignment="Left" Height="100"
        <Image HorizontalAlignment="Left" Height="100"
        <Image HorizontalAlignment="Left" Height="100"
        <Image HorizontalAlignment="Left" Height="100"
        <Image HorizontalAlignment="Left" Height="100"
        <Image HorizontalAlignment="Left" Height="100"
        <TextBox Name="setupBrick" HorizontalAlignment=
        <TextBox Name="setupOre" HorizontalAlignment="L
        <TextBox Name="setupSheep" HorizontalAlignment=
        <TextBox Name="setupWheat" HorizontalAlignment=
        <TextBox Name="setupWood" HorizontalAlignment="
        <TextBox Name="setupGold" HorizontalAlignment="
        <Label Content="Enter the tokens for each hex"
        <Label Content="Please enter only numbers and s
        <Button Name="setupBtn" Content="Done" Horizont
    </Grid>
```

```
1 reference
private void setupBtn_Click(object sender,RoutedEventArgs e) {
    MessageBoxResult result = PauseAndAsk("Are the numbers entered accurate?", "Verify Inputs");
    if (result == MessageBoxResult.No) {
        return;
    }
    int[] brickArray = ParseStringToIntArray(setupBrick.Text);
    int[] goldArray = ParseStringToIntArray(setupGold.Text);
    int[] oreArray = ParseStringToIntArray(setupOre.Text);
    int[] sheepArray = ParseStringToIntArray(setupSheep.Text);
    int[] wheatArray = ParseStringToIntArray(setupWheat.Text);
    int[] woodArray = ParseStringToIntArray(setupWood.Text);
    ////////////////////////////////////////////////////////////////////////////THIS WORKS//////////////////////////////////////////////
    int totalCount = brickArray.Length + goldArray.Length + oreArray.Length + sheepArray.Length + wheatArray.Length + woodArray.Length;
    Location[] allLocations = new Location[totalCount];
    for (int i = 0; i < brickArray.Length; i++) {
        allLocations[i] = ConvertIntToLocation(brickArray[i], "BRICK");
    }
    for (int i = 0; i < goldArray.Length; i++) {
        allLocations[i + brickArray.Length] = ConvertIntToLocation(goldArray[i],"GOLD");
    }
    for (int i = 0; i < oreArray.Length; i++) {
        allLocations[i + brickArray.Length + goldArray.Length] = ConvertIntToLocation(oreArray[i], "ORE");
    }
    for (int i = 0; i < sheepArray.Length; i++) {
        allLocations[i + brickArray.Length + goldArray.Length + oreArray.Length] = ConvertIntToLocation(sheepArray[i], "SHEEP");
    }
    for (int i = 0; i < wheatArray.Length; i++) {
        allLocations[i + brickArray.Length + goldArray.Length + oreArray.Length + sheepArray.Length] = ConvertIntToLocation(wheatArray[i], "WHEAT");
    }
    for (int i = 0; i < woodArray.Length; i++) {
        allLocations[i + brickArray.Length + goldArray.Length + oreArray.Length + sheepArray.Length + wheatArray.Length] = ConvertIntToLocation(woodArray[i], "WOOD");
    }

    mainBoard = new GameBoard(allLocations);
```

Moving to the Main Game Loop

- Buttons to Handle Collection of Dice Roll Data
- Confirmation Button to avoid Accidental Roll Entry

Additionally

- Display Player Turn
- Running Game Timer
- Display Player’s Current Victory Points

11 references

```
private void btnRoll_click(object sender,RoutedEventArgs e) {
    Button resource = sender as Button;
    int roll = int.Parse(resource.Content.ToString());
    selectedRoll = roll;
    SetButtonsUnclicked(rollButtons);
    resource.Background = buttonClicked;
    confirmRoll.Visibility = Visibility.Visible;
}//END METHOD
```

1 reference

```
private void confirmRoll_Click(object sender,RoutedEventArgs e) {
    Label label = null;
    if (selectedRoll != -1) {
        label = GetLabel(selectedRoll);
    }
    if (label != null) {
        mainBoard.AddRoll(selectedRoll);
        label.Content = mainBoard.GetRollCount(selectedRoll);
    }
    SetButtonsUnclicked(rollButtons);
    confirmRoll.Visibility = Visibility.Collapsed;
    DisableRollButtons();
    if (selectedRoll == 7) {
        viewLstBxSelection.Visibility = Visibility.Visible;
        btnKnight.IsEnabled = false;
        btnVicPoint.IsEnabled = false;
        robberActive = true;
        MoveRobber(robberStage);
    } else {
        viewMainLoopPostRollOptions.Visibility = Visibility.Visible;
        lblRobber.Content = UpdatePlayerResources(selectedRoll);
        lblRobber.Visibility = Visibility.Visible;
    }
    selectedRoll = -1;
    btnTurn.IsEnabled = true;
    hasRolled = true;
}//END METHOD
```

Main Loop : After the Dice Roll

- Display Any Resources Distributed to Players
- Enable ‘END TURN’ Option
- Enable Main Turn Options
  - Build Road
  - Build Settlement
  - Build City
  - Buy Development Card


```
1 reference
private string UpdatePlayerResources(int token) {
    string output = "";
    Location[] rolledSpot = mainBoard.GetLocationsByToken(token);
    for (int pl = 0; pl < players.Length; pl++) {
        string holder = $"{players[pl].Name}-";
        for (int loc = 0; loc < rolledSpot.Length; loc++) {
            if (players[pl].IsOnLocation(rolledSpot[loc])) {
                holder += $"{players[pl].UpdateResourceCount(rolledSpot[loc], mainBoard)}";
            }
        }
        if (holder == $"{players[pl].Name}-") {
            holder = "";
        }
        output += holder + "  ";
    }
    return output;
}//END METHOD
```

```
public string UpdateResourceCount(Location target, GameBoard game) {
    int initBrick = _totalBrick;
    int initGold = _totalGold;
    int initOre = _totalOre;
    int initSheep = _totalSheep;
    int initWheat = _totalWheat;
    int initWood = _totalWood;
    int initBlocked = _timesBlocked;
    string blocked = "";

    for (int i = 0; i < _settlementObjects.Length; i++) {
        if (_settlementObjects[i].HasLocation(target) && target == game.Blocked) {
            _timesBlocked += 1;
            blocked = target.Resource;
        } else if (_settlementObjects[i].HasLocation(target)) {
            if (target.Resource == "BRICK") {  _totalBrick += 1;}
            if (target.Resource == "GOLD") { _totalGold += 1;}
            if (target.Resource == "ORE") { _totalOre += 1;}
            if (target.Resource == "SHEEP") { _totalSheep += 1;}
            if (target.Resource == "WHEAT") { _totalWheat += 1;}
            if (target.Resource == "WOOD") { _totalWood += 1;}
        }
    }
    if (_cityObjects != null) {
        for (int i = 0; i < _cityObjects.Length; i++) {
            if (_cityObjects[i].HasLocation(target) && target == game.Blocked) {
                _timesBlocked += 2;
                blocked = target.Resource;
            } else if (_cityObjects[i].HasLocation(target)) {
                if (target.Resource == "BRICK") {  _totalBrick += 2;}
                if (target.Resource == "GOLD") { _totalGold += 2;}
                if (target.Resource == "ORE") { _totalOre += 2;}
                if (target.Resource == "SHEEP") { _totalSheep += 2;}
                if (target.Resource == "WHEAT") { _totalWheat += 2;}
                if (target.Resource == "WOOD") { _totalWood += 2;}
            }
        }
```

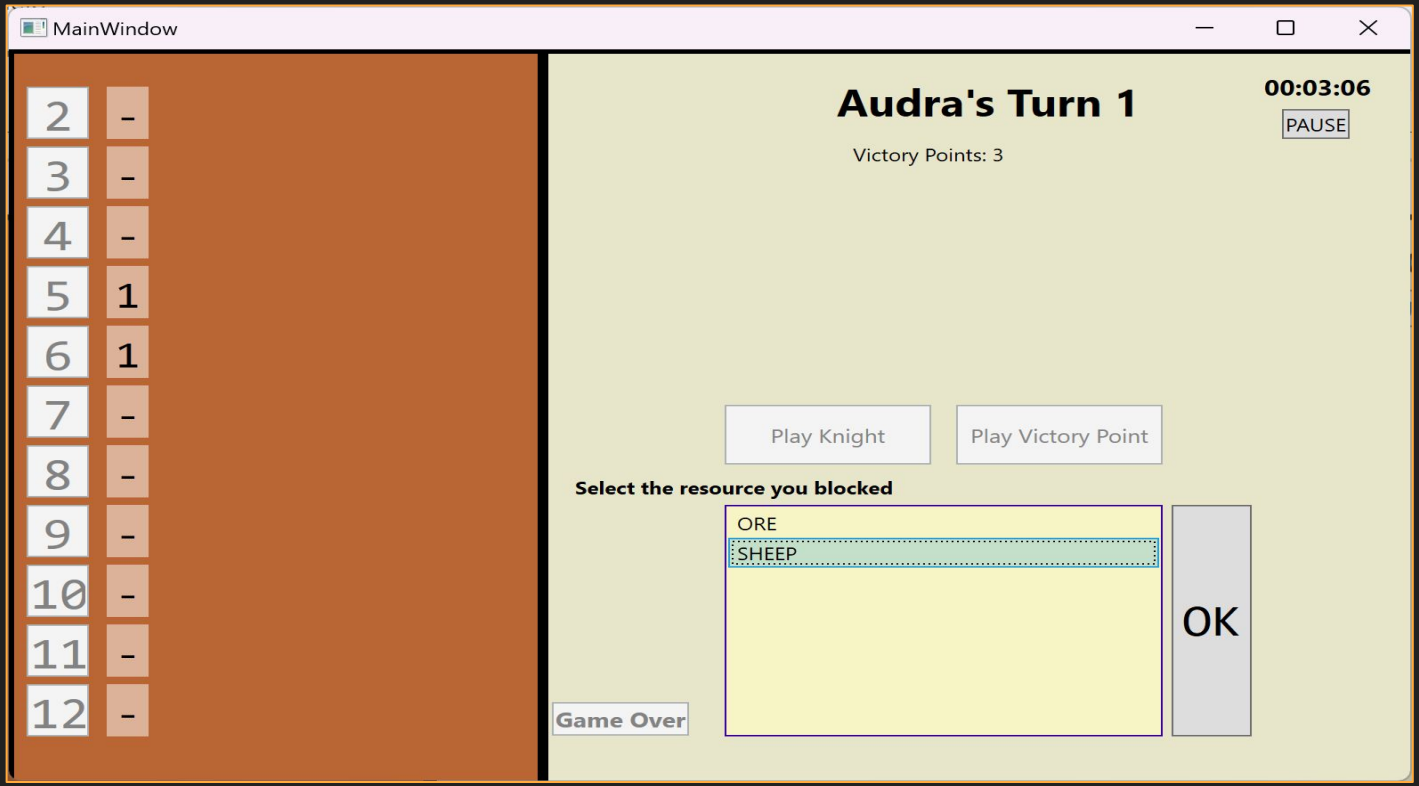
- Iterate Through Players and Check for Ownership of Rolled Number
- If Ownership is Detected
  - Update the Resource Count for Player

Main Turn Actions that can be Taken BEFORE the Roll

BEFORE or AFTER the Dice Roll on Your Turn, You May:

- Play a Knight
  - Moves the Robber onto a New Hex/Token + Steal a Card from Player on that Hex
- Play a Development Card
  - For the purposes of this tracking application, the only tracked option is a Victory Point

On the next screen, we'll see what playing a Knight looks like.



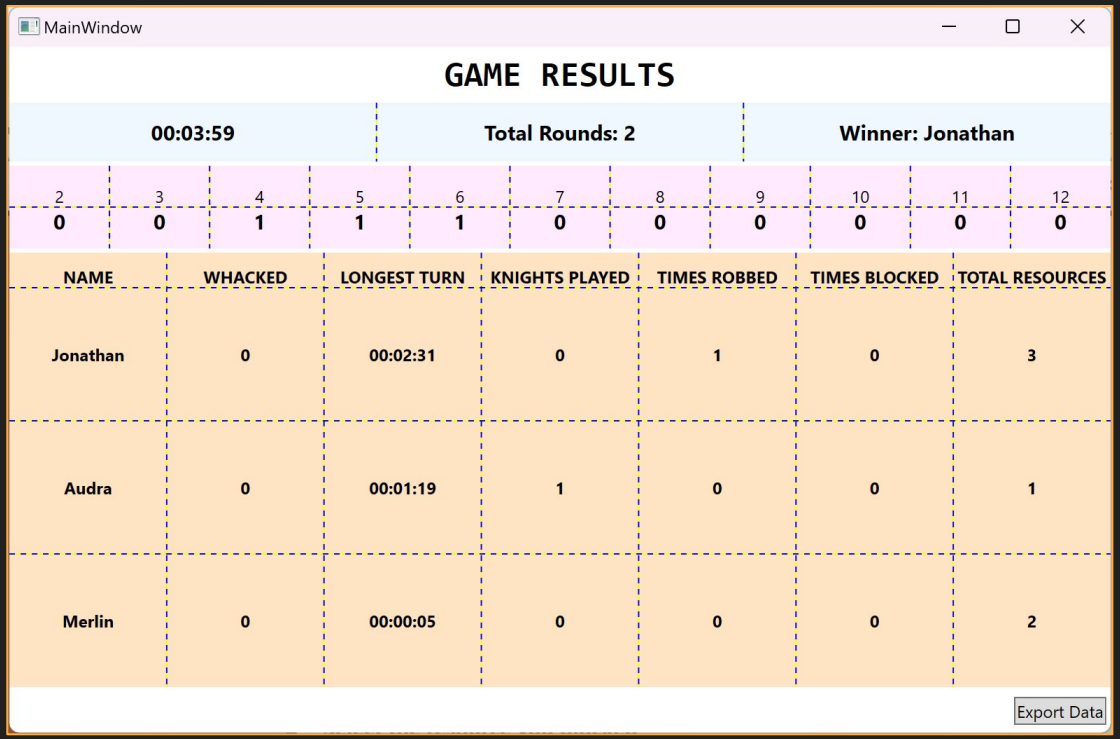
```
int robberStage = 0; // 0 - ASK WHACK ; 1 - CHECK RESPONSE ; 2 - REQUEST TOKEN MOVE ; 3 - REQUEST RESOURCE HIT ; 4 - REQUEST PLAYER HIT ; 5 - RESOLVE HIT
```

MoveRobber() is a LARGE method with multiple phases.
Definitely a target for future code cleanup.

END-GAME

- For Brevity, I'll Forego Showing ALL Functionality of Main Game Loop
  - 27 GameBoard Methods
  - 16 Player Methods
  - 47 MainWindow Methods
  - (Total of 90)

- Get ALL End of Game Data from 'Game' Object and Each 'Player' Object
- Generate XAML Rows/Grids/TextBoxes to Display Data
- Load Data into Displays




```
private void btnGameOver_Click(object sender,RoutedEventArgs e) {
    MessageBoxResult result = PauseAndAsk("Has the game concluded?", "Confirm Game Over");
    if (result == MessageBoxResult.No) {
        return;
    }
    viewMainLoop.Visibility = Visibility.Collapsed;
    viewEndGame.Visibility = Visibility.Visible;
    mainBoard.StopGameTimer();
    TimeSpan turn = mainBoard.GetTurnTime();
    players[turnIndex].AddTurnTime(turn);

    //ASK WHO WON IN A MESSAGE BOX OR SOMETHING
    bool winnerFound = false;
    int index = 0;
    while (!winnerFound) {
        MessageBoxResult winner = PauseAndAsk($"Did {players[index].Name} win?", "SELECT WINNER");
        if (winner == MessageBoxResult.Yes) {
            mainBoard.Winner = players[index];
            players[index].IsWinner = true;
            winnerFound = true;
        } else {
            index += 1;
        }
        index = players.Length == index ? 0 : index;
    }//END WINNER FINDING LOOP

    RowDefinition[] playerStats = new RowDefinition[players.Length];
    for (int i = 0; i < playerStats.Length; i++) {
        playerStats[i] = new RowDefinition();
        grdPlayerResults.RowDefinitions.Add(playerStats[i]);
        players[i].EndStats(grdPlayerResults, i + 1);
    }//END LOOP

    SetEndGameStats(mainBoard);
}//END METHOD
```

Only a handful of data points will fit on this final screen, so I added an ‘EXPORT’ button to allow for long-term data storage including more data points.

EXPORT

Using the StreamWriter and SaveFileDialog classes (from .Net Library) we find where the user wants to save the data.

Using the ‘GetAllGameData()’ method (custom built), we feed one LARGE string (with values separated by commas) into the StreamWriter to save to file.

```
private void btnExport_Click(object sender,RoutedEventArgs e) {
    StreamWriter outfile;
    SaveFileDialog saveFile = new SaveFileDialog();

    saveFile.Filter = "csv files (*.csv)|*.csv";
    saveFile.FilterIndex = 2 ;
    saveFile.RestoreDirectory = true ;

    if (saveFile.ShowDialog() == true) {
        string path = saveFile.FileName;
        outfile = new StreamWriter(path);
        string allData = GetAllGameData();
        for (int i = 0; i < allData.Length; i++) {
            outfile.Write(allData[i]);
        }
        outfile.Close();
    }
}//END METHOD

1 reference
private string GetAllGameData() {
    string data = "";
    data += "GAME STATS\nTOTAL GAME TIME, TOTAL ROUNDS, WINNER\n";
    data += $"{mainBoard.GetGameTime().ToString(@"hh\:mm\:ss")}, {mainBoard.Round}, {mainBoard.Winner.Name}\n\n";
    data += "ROLL DATA\n";
    data += $"2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\n";
    for (int i = 0; i < mainBoard.RollCounter.Length; i++) {
        data += $"{mainBoard.RollCounter[i]}, ";
    }
    data += "\b\b\n\n";
    data += "PLAYER DATA\n";
    //playerData += $"{Name}, {_turnInGame}, {_longestRoadLength}, {_knightCount}, {_stolenFromCount}, {_whackedCo
    data += "Name, Turn Order, Longest Road, Largest Army, Times Robbed, Whacked Count, Wood Earned, Bricks Earned
    for (int i = 0; i < players.Length; i++) {
        data += players[i].GetAllPlayerDataToString();
    }
    return data;
}
```

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	GAME STATS												
2	TOTAL GAME TIME	TOTAL ROUNDS	WINNER										
3	00:03:59	2	Jonathan										
4													
5	ROLL DATA												
6	2	3	4	5	6	7	8	9	10	11	12		
7	0	0	1	1	1	0	0	0	0	0	0		
8													
9	PLAYER DATA												
10	Name	Turn Order	Longest Road	Largest Army	Times Robbed	Whacked Count	Wood Earned	Bricks Earned	Ore Earned	Sheep Earned	Wheat Earned	Gold Earned	Total Resources Earned
11	Jonathan	1	1	0	1	0	0	0	1	2	0	0	3
12	Audra	2	1	1	0	0	0	0	1	0	0	0	1
13	Merlin	3	1	0	0	0	0	1	1	0	0	0	2
14													

Times Blocked	Victory Points	Development Cards Bought	Longest Turn	Shortest Turn	Average Turn	Robber Activity	Knight Activity
0	2	0	00:02:31	00:00:02	00:01:17		Jonathan was knighted by Audra ::
0	3	1	00:01:19	00:01:19	00:01:19		Audra knighted Jonathan ::
0	2	0	00:00:05	00:00:05	00:00:05		

The END-GAME display in the application only shows 7 data categories.

By exporting to a .csv file, I can produce and store as much data as my hard-drive is able to store (such a stupidly large amount that it's not worth calculating).

The above example currently holds 24 data categories with millions of potential room to grow.

Happy Catan Wife!